(4)

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-412

USING MAPPINGS TO PROVE TIMING PROPERTIES

Nancy A. Lynch
Hagit Attiya

September 1989

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TM-412	5 MONITORING ORGANIZATION REPORT NUMBER(S) N00014-85-K-0168, N00014-83-K-0125 and N00014-89-J-1988
6a. NAME OF PERFORMING ORGANIZATION MIT Laboratory for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Department of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Boulevard Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

Using Mappings to Prove Timing Properties

12. PERSONAL AUTHOR(S)

Lynch, N.A. and Attiya, H.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Technical	FROM _____ TO _____	1989 September	44

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Timing properties, timing-based algorithms, formal specification, formal verification, assertional reasoning, possibilities mappings, timed automata, I/O automata

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A new technique for proving timing properties for timing-based algorithms is described; it is an extension of the mapping techniques previously used in proofs of safety properties for asynchronous concurrent systems. The key to the method is a way of representing a system with timing constraints as an automaton whose state includes predictive timing information. Timing assumptions and timing requirements for the system are both represented in this way. A multivalued mapping from the "assumptions automaton" to the "requirements automaton" is then used to show that the given system satisfies the requirements. The technique is illustrated with two simple examples, a resource manager and a signal relay.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little, Publications Coordinator	22b TELEPHONE (Include Area Code) (617) 253-5894 22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆U.S. Government Printing Office: 1988—507-047

Using Mappings to Prove Timing Properties[1]

Nancy A. Lynch and Hagit Attiya

Laboratory for Computer Science
MIT
Cambridge, MA 02139

SEPTEMBER 26, 1989

[1]This work was supported by ONR contract N00014-85-K-0168, by NSF grant CCR-8611442, and by DARPA contracts N00014-83-K-0125 and N00014-89-J-1988

Abstract

A new technique for proving timing properties for timing-based algorithms is described; it is an extension of the mapping techniques previously used in proofs of safety properties for asynchronous concurrent systems. The key to the method is a way of representing a system with timing constraints as an automaton whose state includes predictive timing information. Timing assumptions and timing requirements for the system are both represented in this way. A multivalued mapping from the "assumptions automaton" to the "requirements automaton" is then used to show that the given system satisfies the requirements. The technique is illustrated with two simple examples, a resource manager and a signal relay.

Keywords: Timing properties, timing-based algorithms, formal specification, formal verification, assertional reasoning, possibilities mappings, timed automata, I/O automata.

1 Introduction

Assertional reasoning is a very useful technique for proving safety properties of sequential and concurrent algorithms. This proof method involves describing the algorithm of interest as a state machine, and defining a predicate known as an *assertion* on the states of the machine. One proves inductively that the assertion is true of all the states that are reachable in a computation of the machine, i.e., that it is an *invariant* of the machine. The assertion is defined so that it implies the safety property to be proved.

One kind of assertional reasoning uses a mapping to describe a correspondence between the given algorithm and a higher-level algorithm used as a specification of correctness. (See, for example, [La83, Ly86, LT87].) Such mappings may be single-valued or multivalued.

So far, assertional reasoning has been used primarily to prove properties of sequential algorithms and synchronous and asynchronous concurrent algorithms. It would also be nice to use this technique to prove properties of concurrent algorithms whose operation depends on time, e.g., algorithms that use clocks that tick at approximately predictable rates. Also, the kinds of properties generally proved using assertional reasoning have been "ordinary" safety properties; it would be nice to use similar methods to prove timing properties (upper and lower bounds on time) for algorithms that have timing assumptions. For example, predictable performance is often a desirable characteristic of real-time systems [SR89]; assertional techniques could be very helpful in proving such performance properties.

In this paper, we describe one way in which assertional reasoning can be used to prove timing properties for algorithms that have timing assumptions. Our method involves constructing a multivalued mapping from an automaton representing the given algorithm to another automaton representing the timing requirements. The key to our method is a way of representing a system with timing constraints as an automaton whose state includes predictive timing information. Timing assumptions and timing requirements for the system are both represented in this way, and the mappings we construct map from the "assumptions automaton" to the "requirements automaton".

The formal model we use to describe our method is the *timed automaton* model, a slight variant of the *time constrained automaton* model of [MMT88]. We use this model to state the requirements to be satisfied, to define the basic architectural and timing assumptions, to describe the algorithms, and to prove their correctness and timing properties. A timed automaton is a pair (A, b), consisting of an *I/O automaton* [LT87, LT89], together with a *boundmap*, which is a formal description of the timing assumptions for the components of the system. We introduce the notion of a *timing condition* to state upper and lower bounds on the difference between the times at which certain events or states appear in an execution; the conditions imposed by a boundmap are timing conditions of a particular kind. An automaton and a set of timing conditions, (in particular, a timed automaton) generates a set of *timed executions* and a corresponding set of *timed behaviors*.

While convenient for specifying timing assumptions and requirements, timed automata are not directly suited for carrying out assertional proofs about timing properties, because tim-

ing constraints are described by specially-defined timing conditions rather than being built into the automaton itself. We therefore require a way of incorporating timing conditions into an automaton definition. We do this by means of a general construction of an automaton $time(A,\mathcal{U})$, for a given timed automaton A, and a set $\mathcal{U}$ of timing conditions. The automaton $time(A,\mathcal{U})$ is an ordinary I/O automaton (not a timed automaton) whose state includes predictive information describing the first and last times at which various events can next occur; this information is designed to enforce the timing conditions in $\mathcal{U}$.

In the special case that $\mathcal{U}$ represents the conditions imposed by a boundmap b for A, $time(A,\mathcal{U})$ is the automaton $time(A)$ defined in [AtL89]; this is denoted by $time(A,b)$ in this paper. The I/O automaton $time(A,b)$ is related to the timed automaton (A,b) in that a certain subset of the behaviors of $time(A,b)$, the "complete" behaviors, is exactly equal to the set of timed behaviors of (A,b).

The timing requirements to be proved for an algorithm described as a timed automaton, (A,b), are described as a set of timing conditions, $\mathcal{U}$, for A. We define the *requirements automaton* to be $time(A,\mathcal{U})$. Thus, we build into the state of the requirements automaton predictive information about the first and last times at which certain events of interest can next occur.

The problem of showing that a given algorithm (A,b) satisfies the timing requirements is then reduced to that of showing that any behavior of the automaton $time(A,b)$ is also a behavior of $time(A,\mathcal{U})$. We do this by using invariant assertion techniques; in particular, we demonstrate a multivalued mapping from $time(A,b)$ to $time(A,\mathcal{U})$.

In order to demonstrate the use of our technique, we apply them to two simple examples. The first example is a timing-dependent system consisting of two concurrently-operating components, which we call a *clock* and a *manager*. The clock ticks at an approximately known rate. The manager monitors the clock ticks, and after a certain number have occurred, it issues a $GRANT$ (of a resource). It then continues counting ticks; whenever sufficiently many have occurred since the previous $GRANT$ event, the manager issues another $GRANT$. We give careful proofs of upper and lower bounds on the amount of time prior to the first $GRANT$ event and in between each successive pair of $GRANT$ events.

The second example is an asynchronous (not timing-dependent) system consisting of a "line" of processes. Each process waits to receive a $SIGNAL$ from the process at its left and then sends a $SIGNAL$ to the process at its right. We give careful proofs of upper and lower bounds on the time to propagate a $SIGNAL$ from the left end to the right end of the line. Both of these examples are extremely simple; however, the ideas they embody also appear in many more realistic examples.

The mappings we provide for both of these examples have a particularly interesting and simple form -- a set of inequalities relating the time bounds to be proved to those that can be computed from the state. These inequalities contain information about how the bounds are to be satisfied.

Another interesting aspect of the second example is that the proof is carried out using a

hierarchy of automata, rather than just a pair of automata; the given system is the lowest level, and the requirements automaton is the highest level in the hierarchy. We define a mapping for each level in the hierarchy; the composition of the entire collection of mappings is the mapping needed to show correctness. The hierarchical proof is especially interesting because its assertional reasoning corresponds closely to the more "operational" reasoning that might be used in an alternative proof based on recurrences.

Technically, mapping techniques of the sort used in this paper are only capable of proving safety properties, but not liveness properties. Timing properties have aspects of both safety and liveness. A timing lower bound asserts that an event cannot occur before a certain amount of time has elapsed; a violation of this property is detectable after a finite prefix of a timed execution, and so a timing lower bound can be regarded as a safety property. A timing upper bound asserts that an event must occur before a certain amount of time has elapsed. This can be regarded as making two separate claims: that the designated amount of time does in fact elapse (a liveness property), and that that time cannot elapse without the event having occurred (a safety property). In this paper, we assume the liveness property that time increases without bound, so that all the remaining properties that need to be proved in order to prove either upper or lower time bounds are safety properties. Thus, our mapping technique provides complete proofs for timing properties without requiring any special techniques (e.g., variant functions or temporal logic methods) for arguing liveness.

There has been some prior work on using assertional reasoning to prove timing properties. In particular, Haase [H81], Shankar and Lam [SL87], Tel [T88], Schneider [S88], Lewis [H89] and Shaw [S89] have all developed models for timing-based systems that incorporate time information into the state, and have used invariant assertions to prove timing properties. In [T88] and [H89], in fact, the information that is included is similar to ours in that it is also predictive timing information (but not exactly the same information as ours). None of this work has been based on mappings, however.

Several other, quite different formal approaches to proving timing properties have also been developed. Some representative papers describing these other methods are [BH81], [KVR83], [JM87], [Ho87], [Zw88], [JS88], and [GF88].

The rest of the paper is organized as follows. Section 2 contains a description of the underlying formal models: I/O automata, timed automata and timing conditions. Section 3 contains the general construction used to produce the $time(A,\mathcal{U})$ automata, and some properties of these automata. Section 4 contains our first example, a simple resource-granting manager using a clock; the section contains a description of the algorithm, a description of the corresponding requirements automaton, and a correctness proof. Section 5 contains a similar treatment for our second example – a simple signal propagation system. We conclude with a discussion in Section 6. Some of the more technical proofs are relegated to an Appendix.

2 Formal Model

In this section, we present the definitions for the underlying formal model. In particular, we define I/O automata, timed automata and timing conditions. We also present some of their relevant properties.

2.1 I/O Automata

We begin by summarizing some of the key definitions for the I/O automaton model. We refer the reader to [LT87, LT89] for a complete presentation of the model and its properties.

An *I/O automaton* consists of the following components: $acts(A)$, a set of *actions*, classified as *output*, *input* and *internal* (input and output actions are called *external*); $states(A)$, a set of *states*, including a distinguished subset, $start(A)$, of *start states*; $steps(A)$, a set of steps, where a *step* is defined to be a (*state*, *action*, *state*) triple; and $part(A)$, a *partition* of the locally controlled (output and internal) actions into equivalence classes; the partition groups together actions that are to be thought of as under the control of the same underlying process.

An action π is said to be *enabled* in a state s' provided that there is a step of the form (s', π, s). An automaton is required to be *input enabled*, which means that every input action must be enabled in every state. For any set $\Pi \subseteq acts(A)$, we denote by $enabled(A, \Pi)$ the set of states of A in which some action in Π is enabled, and by $disabled(A, \Pi)$ be the set of all states of A not in $enabled(A, \Pi)$, that is, $disabled(A, \Pi) = states(A) \setminus enabled(A, \Pi)$.

An *execution* of an I/O automaton A is a sequence (finite or infinite) of alternating states and actions

$$s_0, \pi_1, s_1, \ldots, s_{i+1}, \pi_i, s_i, \ldots$$

where $s_0 \in init(A)$, and for every i, $(s_{i-1}, \pi_i, s_i) \in steps(A)$. (If the sequence is finite, then it is required to end with a state.) The *schedule* of an execution α is the subsequence consisting of the actions appearing in α and is denoted $sched(\alpha)$. The *behavior* of an execution α of A is the subsequence of α consisting of external actions appearing in α and is denoted $beh(\alpha)$. The *schedules* and *behaviors* of A are just those of the executions of A.

Concurrent systems are modeled by compositions of I/O automata, as defined in [LT87, LT89]. In order to be composed, automata must be *strongly compatible*; this means that no action can be an output of more than one component, that internal actions of one component are not shared by any other component, and that no action is shared by infinitely many components. The result of such a composition is another I/O automaton. The *hiding* operator can be applied to reclassify output actions as internal actions.

We now define the kind of mapping we will use to describe correspondences between I/O automata. The definition we use here is a slight strengthening of the "possibilities mapping" definition of [LT87, LT89], designed to preserve the entire sequence of actions, internal as well as external, produced by the automaton.

Definition 2.1 *Let A and B be I/O automata with the same external actions, and let f be a mapping from states of A to sets of states of B. The mapping f is a* strong possibilities mapping from A to B *provided that the following conditions hold:*

1. *For every start state s_0 of A, there is a start state u_0 of B such that $u_0 \in f(s_0)$.*
2. *If s' is a reachable state of A, $u' \in f(s')$ is a reachable state of B, and (s', π, s) is a step of A, then there is a step (u', π, u) of B such that $u \in f(s)$.*

Lemma 2.1 *If there exists a strong possibilities mapping from A to B, then every schedule of A is also a schedule of B.*

Proof: Straightforward. ■

2.2 Timed Automata

In this subsection, we augment the I/O automaton model to allow discussion of timing assumptions. The treatment here is the same as described in [AtL89] and is a special case of the definitions proposed earlier in [MMT88]).

A *boundmap* for an I/O automaton A is a a mapping that associates a closed subinterval of $[0, \infty]$ with each class in $part(A)$, where the lower bound of each interval is not ∞ and the upper bound is nonzero. Intuitively, the interval associated with a class C by the boundmap represents the range of possible lengths of time between successive times when C "gets a chance" to perform an action. We sometimes use the notation $b_\ell(C)$ to denote the lower bound assigned by boundmap b to class C, and $b_u(C)$ for the corresponding upper bound. A *timed automaton* is a pair (A, b), where A is an I/O automaton and b is a boundmap for A.

We require notions of "timed execution", "timed schedule" and "timed behavior" for timed automata, corresponding to executions, schedules and behaviors for ordinary I/O automata. These will all include time components. We begin by defining the basic type of sequence that underlies the definition of a timed execution.

A *timed sequence* is a (finite or infinite) sequence of alternating states and (action,time) pairs,

$$s_0, (\pi_1, t_1), s_1, (\pi_2, t_2), \ldots ,$$

ending in a state if the sequence is finite. Define $t_0 = 0$. The times $t_0, t_1, \ldots$ are required to be nondecreasing, and if the sequence is infinite then the times are also required to be unbounded. For any finite timed sequence α define $t_{end}(\alpha)$ to be the time of the last event in α, if α contains any (action,time) pairs, or 0, if α contains no such pairs; also, define $s_{end}(\alpha)$ to be the last state in α. We denote by $ord(\alpha)$ (the "ordinary" part of α) the sequence

$$s_0, \pi_1, s_1, \pi_2, \ldots ,$$

i.e., α with time components removed.

Definition 2.2 *Suppose* (A, b) *is a timed automaton. Then a timed sequence* α *is a* timed execution *of* (A, b) *provided that* $ord(\alpha)$ *is an execution of* A *and* α *satisfies the following conditions, for each class* $C \in part(A)$ *and every* i.

1. *Suppose* $b_u(C) < \infty$. *If* $s_i \in enabled(A, C)$ *and either* $i = 0$ *or* $s_{i-1} \in disabled(A, C)$ *or* $\pi_i \in C$, *then there exists* $j > i$ *with* $t_j \leq t_i + b_u(C)$ *such that either* $\pi_j \in C$ *or* $s_j \in disabled(A, C)$.

2. *If* $s_i \in enabled(A, C)$ *and either* $i = 0$ *or* $s_{i-1} \in disabled(A, C)$ *or* $\pi_i \in C$, *then there does not exist* $j > i$ *with* $t_j < t_i + b_\ell(C)$ *and* π_j *in* C.

The first condition says that, starting from when an action in C occurs or first gets enabled, within time $b_u(C)$ either some action in C occurs or there is a point at which no such action is enabled. Note that if $b_u(C) = \infty$, no upper bound requirement is imposed. The second condition says that, again starting from when an action in C occurs or first gets enabled, no action in C can occur before time $b_\ell(C)$ has elapsed.

The *timed schedule* of a timed execution of a timed automaton (A, b) is the subsequence consisting of the (action,time) pairs, and the *timed behavior* is the subsequence consisting of the (action,time) pairs for which the action is external. The *timed schedules* and *timed behaviors* of (A, b) are just those of the timed executions of (A, b). The following lemma is a straightforward result of the definition of a timed execution.

Lemma 2.2 *Suppose that* α *is a finite timed execution of a timed automaton* (A, b). *Then each locally controlled action of* A *that is enabled in the final state of* $s_{end}(\alpha)$ *is in a partition class* C *in* $part(A)$ *such that* $b_u(C) = \infty$.

We model each timing-dependent concurrent system as a single timed automaton (A, b), where A is a composition of ordinary I/O automata (possibly with some output actions hidden).[1]

We give one more definition here, for use in later proofs. Notice that the definition of a timed execution contains aspects of both safety and liveness. Sometimes it it useful to focus on the safety aspects alone. We thus define the notion of a "timed semi-execution" to capture the safety part of the definition of a timed execution.

Definition 2.3 *Suppose* (A, b) *is a timed automaton. Then a finite timed sequence* α *is a* timed semi-execution *of* (A, b) *provided that* $ord(\alpha)$ *is an execution of* A *and* α *satisfies the following conditions, for each class* C *of* $part(A)$ *and every* i.

[1] An equivalent way of looking at each system is as a composition of timed automata. An appropriate definition for a composition of timed automata is developed in [MMT88], together with theorems showing the equivalence of the two viewpoints.

1. *Suppose* $b_u(C) < \infty$. *If* $s_i \in enabled(A, C)$ *and either* $i = 0$ *or* $s_{i-1} \in disabled(A, C)$ *or* $\pi_i \in C$, *then either* $t_{end}(\alpha) \leq t_i + b_u(C)$ *or there exists* $j > i$ *with* $t_j \leq t_i + b_u(C)$ *such that either* $\pi_j \in C$ *or* $s_j \in disabled(A, C)$.

2. *If* $s_i \in enabled(A, C)$ *and either* $i = 0$ *or* $s_{i-1} \in disabled(A, C)$ *or* $\pi_i \in C$, *then there does not exist* $j > i$ *with* $t_j < t_i + b_\ell(C)$ *and* $\pi_j \in C$.

This definition is identical to that of a timed execution, except for the (first) "either" clause in the first item. This clause allows an action to fail to occur if insufficient time has passed.

2.3 Timing Conditions

The conditions imposed by a boundmap are appropriate for describing the timing assumptions of many systems. However, in order to describe the timing requirements that are to be proved for these systems, it is convenient to generalize these conditions. For example, a bound is often required on the time between two particular events, e.g., a request and a corresponding grant. It is convenient to have a notation that permits explicit description of such a condition, without reference to the underlying partition classes. Therefore, in this subsection, we generalize the conditions expressed by boundmaps to more general "timing conditions".

Let A be an I/O automaton. A *timing condition* for A is a tuple of the form $(T_{start}, T_{step}, b, \Pi, S)$, where:

- $T_{start} \subseteq start(A)$ and $T_{step} \subseteq steps(A)$, are the *triggers*.
- b is a closed interval of the form $[b_\ell, b_u]$, where $b_\ell \neq \infty$ and $b_u \neq 0$,
- $\Pi \subseteq acts(A)$, and
- $S \subseteq states(A)$ is the *disabling set*.

We require that a timing condition satisfy the following technical conditions:

1. $T_{start} \cap S = \emptyset$, and
2. if $(s', \pi, s) \in T_{step}$ then $s \notin D$.

A timing condition $(T_{start}, T_{step}, b, \Pi, S)$ is designed to specify upper and lower bounds on the time until the next occurrence of an event in the action set Π, measured from certain points during an execution; the particular bounds are given by the interval b. The trigger T_{start} specifies those start states from which we wish to begin measuring the time, while the trigger T_{step} specifies those steps after which we wish to begin measuring. In both cases, the numerical bounds are the same.

Primarily because we wish this generalization to include conditions imposed by boundmaps as a special case, we must include a way of disabling the bound measurements. (In the case of boundmaps, when all the actions in a partition class become disabled simultaneously, the bound measurement also becomes disabled.) Thus, the disabling set S is defined to be a set of states that cause the bound measurement to become suspended. Conditions 1. and 2. simply say that the disabling set does not include any states that the triggers designate as states in which to start the bound measurement.

We sometimes write the timing condition $(T_{start}, T_{step}, b, \Pi, S)$ in the form

$$(T_{start}, T_{step}) \overset{b}{\rightsquigarrow} (\Pi, S).$$

A timing condition can be used to specify only a lower bound or only an upper bound, by making the other bound trivial (0 for lower bounds, ∞ for upper bounds).

Now we define what it means for a timed sequence to satisfy a timing condition. The definition is closely related to the definition we gave earlier of a timed execution; we will show the precise connection in Lemma 2.3.

Definition 2.4 *Let α be the timed sequence $s_0, (\pi_1, t_1), s_1, \ldots$. Then α* satisfies *a timing condition $(T_{start}, T_{step}) \overset{b}{\rightsquigarrow} (\Pi, S)$ if the following conditions hold:*

1. *Suppose $b_u < \infty$.*
   - *(a) If $s_0 \in T_{start}$ then there exists $j > 0$ with $t_j \leq b_u$ such that either $\pi_j \in \Pi$ or $s_j \in D$.*
   - *(b) If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ then there exists $j > i$ with $t_j \leq t_i + b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
2. *(a) If $s_0 \in T_{start}$ and if there exists $j > 0$ with $t_j < b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, 0 < k < j$, such that $s_k \in S$.*
   - *(b) If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ and if there exists $j > i$ with $t_j < t_i + b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, i < k < j$, such that $s_k \in S$.*

Let $\mathcal{U}$ be a set of timing conditions for an I/O automaton A. We say that a timed sequence α is a *timed execution* of $(A, \mathcal{U})$ provided that $ord(\alpha)$ is an execution of A and α satisfies every timing condition $U \in \mathcal{U}$.

To justify this new use of the term "timed execution", and as an example of the use of timing conditions, we show how to express the notion of "timed execution" of (A, b) in terms of timing conditions. Given an arbitrary timed automaton (A, b), we define the set $\mathcal{U}_b$ of timing conditions that are *associated with* b. For each class C in the partition of A, $\mathcal{U}_b$ includes one timing condition, $cond(C) = (T_{start}(C), T_{step}(C)) \overset{b(C)}{\rightsquigarrow} (\Pi(C), S(C))$, defined as follows.

- $T_{start}(C) = start(A) \cap enabled(A, C)$, that is, the set of start states of A in which some action from C is enabled,

- $T_{step}(C)$ is the set of steps (s', π, s) of A such that $s \in enabled(A, C)$ and either $s' \in disabled(A, C)$ or $\pi \in C$,
- $\Pi(C) = C$, and
- $S(C) = disabled(A, C)$.

Note that this definition satisfies the two requirements for timing conditions.

Lemma 2.3 *Suppose (A, b) is a timed automaton. Let α be a timed sequence and suppose that $ord(\alpha)$ is an execution of A. Then the following two statements are equivalent.*

1. *α is a timed execution of (A, b).*
2. *For every class $C \in part(A)$, α satisfies the timing condition $cond(C)$.*

Proof: Let

$$\alpha = s_0, (\pi_1, t_1), s_1, \ldots$$

be a timed sequence such that $ord(\alpha)$ is an execution of A. First assume that α is a timed execution of (A, b). Let $C \in part(A)$; we show that α satisfies $cond(C)$. The upper bound is a simple substitution. For the lower bound we check only triggering start states, the case of triggering steps is similar. If $s_0 \in T_{start}(C)$, then $s_0 \in enabled(A, C)$. Assume that $\pi_j \in C$, for some $j > 0$. Then from Condition 2. of Definition 2.2 it follows that $t_j \geq b_\ell(C)$, which suffices.

Now assume that α satisfies $cond(C)$ for each $C \in part(C)$; we show that α is a timed execution of (A, b). Again, the upper bound holds easily and the only interesting case to verify is the lower bound. Assume, by way of contradiction, that for some class $C \in part(A)$, there exists an $i \geq 0$, such that $s_i \in enabled(A, C)$ and either $i = 0$ or $s_{i-1} \in disabled(A, C)$ or $\pi_i \in C$, and that there exists $j > i$, such that $t_j < t_i + b_\ell(C)$ and $\pi_j \in C$. Since α satisfies $cond(C)$, and since $S(C) = disabled(A, C)$, it follows that there is some k, $i < k < j$, such that $s_k \in disabled(A, C)$. Let k_0 be the largest such k. But then $(s_{k_0}, \pi, s_{k_0+1}) \in T_{step}(C)$, $t_j < t_{k_0+1} + b_\ell(C)$ and there is no k', $k_0 < k' < j$ such that $s_{k'} \in S(C)$; this contradicts the fact that α satisfies $cond(C)$. ■

Lemma 2.3 implies the following corollary.

Corollary 2.4 *Suppose (A, b) is a timed automaton. Then a timed sequence α is a timed execution of (A, b) if and only if it is a timed execution of $(A, \mathcal{U}_b)$.*

We note that the definition we use for timing condition may not be the most general condition needed to capture all interesting timing requirements. It does capture many, however; we will have more to say about this matter in the conclusions section.

2.4 Timed Semi-Executions

Recall that the definition of a timed semi-execution of a timed automaton captured the safety portion of the timed execution definition. We can also restrict attention to the safety portions of some of the definitions of the previous subsection, as follows.

Definition 2.5 *Let α be the finite timed sequence $s_0, (\pi_1, t_1), s_1, \ldots, s_{end}$. Then α* semi-satisfies *a timing condition $(T_{start}, T_{step}) \overset{b}{\rightsquigarrow} (\Pi, S)$ if the following conditions hold:*

1. *Suppose $b_u < \infty$.*
   (a) *If $s_0 \in T_{start}$ then either $t_{end}(\alpha) \leq b_u$ or there exists $j > 0$ with $t_j \leq b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
   (b) *If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ then either $t_{end}(\alpha) \leq t_i + b_u$ or there exists $j > i$ with $t_j \leq t_i + b_u$ such that either $\pi_j \in \Pi$ or $s_j \in S$.*
2. (a) *If $s_0 \in T_{start}$ and if there exists $j > 0$ with $t_j < b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, 0 < k < j$, such that $s_k \in S$.*
   (b) *If $(s_{i-1}, \pi_i, s_i) \in T_{step}$ and if there exists $j > i$ with $t_j < t_i + b_\ell$ such that $\pi_j \in \Pi$, then there exists $k, i < k < j$, such that $s_k \in S$.*

Once again, the only differences between this definition and Definition 2.4 are the "either" clauses. Now suppose $\mathcal{U}$ is a set of timing conditions for an I/O automaton A. A timed sequence α is a *timed semi-execution* of $(A, \mathcal{U})$ if $ord(\alpha)$ is an execution of A and for any timing condition $U \in \mathcal{U}$, α semi-satisfies U. Similar results to those for timed executions hold for timed semi-executions.

Lemma 2.5 *Let (A, b) be a timed automaton. Let α be a timed sequence and suppose that $ord(\alpha)$ is a prefix of an execution of A. Then the following two statements are equivalent.*

1. *α is a timed semi-execution of (A, b).*
2. *For every class $C \in part(A)$, α semi-satisfies the timing condition $cond(C)$.*

Corollary 2.6 *For any I/O automaton A and for any boundmap, b, for A, a timed sequence α is a timed semi-execution of (A, b) if and only if it is a timed semi-execution of $(A, \mathcal{U}_b)$.*

Another observation we use later is the following, saying that the limit of a sequence of timed semi-executions in which the time components are unbounded must be a timed execution.

Lemma 2.7 *Let $\{\alpha_i\}_{i=1}^{\infty}$ be a sequence of timed semi-executions of $(A, \mathcal{U})$ such that*

1. *for any* $i \geq 1$, α_i *is a prefix of* α_{i+1}, *and*

2. $\lim_{i \to \infty} t_{end}(\alpha_i) = \infty$.

Then there exists a unique infinite timed execution α *of* $(A,\mathcal{U})$ *such that for any* $i \geq 1$, α_i *is a prefix of* α.

Proof: Straightforward. ∎

3 Incorporating Timing Conditions into I/O Automata

In order to use invariant assertion techniques to reason about timed automata, we define an ordinary I/O automaton $time(A,\mathcal{U})$ corresponding to a given timed automaton A with timing conditions $\mathcal{U}$. This new automaton has the timing restrictions imposed by $\mathcal{U}$ on A built into its transition rules, based on predictions about when the next event from each set of actions will occur. In this section, we give the construction of $time(A,\mathcal{U})$ and also give results relating the executions and behaviors of $time(A,\mathcal{U})$ to the timed executions and timed behaviors of $(A,\mathcal{U})$.

A special and important example of this construction is when $\mathcal{U}$ is the set of conditions corresponding to a boundmap for A, i.e., $\mathcal{U}_b$. In this case, we denote the automaton by $time(A,b)$.[2] In order to provide a concrete example of the construction we present an explicit description of $time(A,b)$ in Section 3.2. We also give some additional results that can be proved for $time(A,b)$ because of its special structure. Other special cases of the general construction will be the requirements automata for the two examples we consider in Sections 4 and 5.

3.1 The General Construction

Given any I/O automaton A and set $\mathcal{U}$ of timing conditions for A, we define the ordinary I/O automaton $time(A,\mathcal{U})$ as follows. We write each timing condition $U \in \mathcal{U}$ as

$$(T_{start}(U), T_{step}(U)) \overset{b(U)}{\rightsquigarrow} (\Pi(U), S(U)) .$$

The automaton $time(A,\mathcal{U})$ has actions of the form (π,t), where π is an action of A and t is a nonnegative real number, with the classification of actions the same as for A. Each of its states consists of a state, As, of A (the "A-state"), augmented with a component Ct (the "current time"), and, for each timing condition $U \in \mathcal{U}$, two components $Ft(U)$ and $Lt(U)$ (the "first time" and "last time" for each timing condition). Ct represents the time of the last preceding event. The $Ft(U)$ and $Lt(U)$ components represent, respectively, the first and last times at which the timing condition U specifies that an action in $\Pi(U)$ should occur.

[2] This automaton was denoted $time(A)$ in [AtL89].

We use record notation to denote the various components of the state of $time(A,\mathcal{U})$: for instance, $s.As$ denotes the state of A included in state s of $time(A,\mathcal{U})$. Each initial state of $time(A,\mathcal{U})$ consists of an initial state s of A, plus $Ct = 0$, plus values of $Ft(U)$ and $Lt(U)$ with the following property: if $s.As \in T_{start}(U)$ then $s.Ft(U) = b_\ell(U)$ and $s.Lt(U) = b_u(U)$; otherwise, $s.Ft(U) = 0$ and $s.Lt(U) = \infty$. That is, if the start state of A is in the trigger set of U, then the predicted times are the ones specified in U; otherwise, they are set to default values.

If (π,t) is an action of $time(A,\mathcal{U})$, then $(s',(\pi,t),s)$ is a step of $time(A,\mathcal{U})$ exactly if the following conditions hold.

1. $(s'.As,\pi,s.As)$ is a step of A.
2. $s'.Ct \leq t = s.Ct$.
3. For all $U \in \mathcal{U}$, if $\pi \in \Pi(U)$, then
   (a) $s'.Ft(U) \leq t \leq s'.Lt(U)$.
   (b) if $(s'.As,\pi,s.As) \in T_{step}(U)$ then $s.Ft(U) = t + b_\ell(U)$ and $s.Lt(U) = t + b_u(U)$,
   (c) if $(s'.As,\pi,s.As) \notin T_{step}(U)$ then $s.Ft(U) = 0$ and $s.Lt(U) = \infty$.
4. For all $U \in \mathcal{U}$, if $\pi \notin \Pi(U)$, then
   (a) $t \leq s'.Lt(U)$,
   (b) if $(s'.As,\pi,s.As) \in T_{step}(U)$ then $s.Ft(U) = t+b_\ell(U)$ and $s.Lt(U) = min(s'.Lt(U), t+b_u(U)$, and
   (c) if $(s'.As,\pi,s.As) \notin T_{step}(U)$ and $s.As \notin S(U)$ then $s.Ft(U) = s'.Ft(U)$ and $s.Lt(U) = s'.Lt(U)$, and
   (d) if $s.As \in S(U)$ then $s.Ft(U) = 0$ and $s.Lt(U) = \infty$.

Intuitively, Condition 1. says that the automaton $time(A,\mathcal{U})$ is correctly simulating the state transitions of A, and Condition 2. says that Ct components are monotonically nondecreasing, i.e., the new time is at least as great as the previous time. Condition 3. deals with properties involving timing conditions U that include π in their action sets: Condition 3(a) says that the time at which the event π occurs must be between the bounds specified for U; Condition 3(b) says that a triggering step involving π imposes new time predictions for U, whereas Condition 3(c) says that a non-triggering step involving π does not impose any such predictions. Condition 4. deals with properties involving timing conditions U that do not include π in their action sets: Condition 4(a) says that π can only occur if U does not require any other action to be scheduled first. Condition 4(b) says that a triggering step involving π imposes new time predictions for U. Note that in this case, there may already be old predictions in effect for this time condition; the effect of taking the min for the $Lt(U)$ component

is to require both the new predictions and any old predictions to be satisfied.[3] Condition 4(c) says that a non-triggering (and non-disabling) step involving π does not impose any new time predictions for U. Condition 4(d) says that a disabling step involving π sets the time predictions for U back to their defaults.

The partition classes for $time(A,\mathcal{U})$ are derived one-for-one from those of A.[4]

We now relate the timed executions of $(A,\mathcal{U})$ to the executions of the corresponding I/O automaton $time(A,\mathcal{U})$. If α is an execution of $time(A,\mathcal{U})$, we define $project(\alpha)$ to be the timed sequence obtained from α by mapping each occurrence of a state s in α to $s.As$ (while keeping the (action,time) pairs intact). We first show the following simple correspondence between semi-executions of $(A,\mathcal{U})$ and finite executions of $time(A,\mathcal{U})$.

Lemma 3.1 *1. If α' is a timed semi-execution of $(A,\mathcal{U})$, then there exists an execution α of $time(A,\mathcal{U})$ such that $\alpha' = project(\alpha)$.*

2. If α is a finite execution of $time(A,\mathcal{U})$, then $project(\alpha)$ is a timed semi-execution of $(A,\mathcal{U})$.

Proof: 1. Suppose that α' is a given timed semi-execution of $(A,\mathcal{U})$. Then there is a unique timed sequence α whose states are states of $time(A,\mathcal{U})$, that has $\alpha' = project(\alpha)$, whose initial state is the unique start state of $time(A,\mathcal{U})$, and each of whose steps satisfies Conditions 1, 3(b), 3(c), 4(b), 4(c) and 4(d) of the definition of $time(A,\mathcal{U})$, plus the equality part of Condition 2. of the definition of $time(A,\mathcal{U})$. The fact that α' is a timed sequence in which, by definition, the time components are non-decreasing, implies the inequality part of 2. Condition 2. of Definition 2.5 ensures the lower bound part of 3(a) of the definition of $time(A,\mathcal{U})$, while Condition 3. of Definition 2.5 ensures the upper bound part of 3(a) and also 4(a) of the definition of $time(A,\mathcal{U})$.

2. By Condition 1. of the definition of $time(A,\mathcal{U})$, $ord(project(\alpha)$ is an execution of the ordinary I/O automaton A. It remains to show that for every timing condition $U \in \mathcal{U}$, $project(\alpha)$ semi-satisfies U. The initialization and Conditions 3(a) and 4(a) of the definition of $time(A,\mathcal{U})$ ensure property 1(a) of Definition 2.5. Conditions 3(b), 4(b), 3(a) and 4(a) of the definition of $time(A,\mathcal{U})$ ensure property 1(b) of Definition 2.5. The initialization and Condition 3(a) of the definition of $time(A,\mathcal{U})$ ensure property 2(a) of Definition 2.5, while Conditions 3(b), 4(b), 3(a) and 4(a) ensure property 2(b) of Definition 2.5.

■

[3] The min is necessary because in case there is a prior prediction, it will surely be no greater than the new prediction, so the min will be the first term $s'.Lt(U)$. However, if there is no prior prediction then $s'.Lt(U) = \infty$ so the min will be the second term $t+b_u(U)$. We could have similarly written $s.Ft(U) = max(s'.Ft(U), t+b_\ell(U))$, but that is unnecessary because it is always the case that $s'.Ft(U) \leq b_\ell(U)$.

[4] It seems that we never need them, however, since the partition classes are used to enforce fairness to the components of the system; in $time(A,\mathcal{U})$ the timing conditions guarantee that each component gets a fair chance to operate.

We give a similar result for infinite sequences:

Lemma 3.2 *1. If α' is an infinite timed execution of $(A,\mathcal{U})$, then there exists an infinite execution α of $time(A,\mathcal{U})$ in which the time components of the actions are unbounded, such that $\alpha' = project(\alpha)$.*

2. If α is an infinite execution of $time(A,\mathcal{U})$ in which the time components of the actions are unbounded, then $project(\alpha)$ is a timed execution of $(A,\mathcal{U})$.

Proof: 1. By the same reasoning as for part 1. of Lemma 3.1; the time components are unbounded since α' is an infinite timed sequence.

2. Let $\alpha = s_0,(\pi_1,t_1),s_1,\ldots$ and let $\alpha_i = s_0,(\pi_1,t_1),\ldots,s_i$, for all $i \geq 0$. Since α_i is a finite execution of $time(A,\mathcal{U})$, $\alpha'_i = project(\alpha_i)$ is a timed semi-execution of $(A,\mathcal{U})$, by part (2) of Lemma 3.1. Since the time components of the actions in α are unbounded, it follows that $\lim_{i\to\infty} t_{end}(\alpha'_i) = \infty$. Lemma 2.7 implies that $project(\alpha)$ is a timed execution of $(A,\mathcal{U})$.

■

3.2 Special Case: The Automaton $time(A,b)$

A very important special case of the construction described in the previous subsection is the case of $time(A,\mathcal{U}_b)$; this automaton is the result of incorporating the boundmap timing conditions of a timed automaton (A,b) into the automaton transitions. As shorthand, we will sometimes refer to this automaton as $time(A,b)$ instead of $time(A,\mathcal{U}_b)$, suppressing explicit mention of the timing conditions $\mathcal{U}_b$. We will also sometimes write $Ft(C)$ instead of $Ft(cond(C))$ for partition class C, and similarly for the other state components.

Because the conditions imposed by a boundmap are fundamental and common instances of timing conditions, and in order to provide an example to illustrate the $time(A,\mathcal{U})$ definition, we now give an explicit definition of $time(A,b)$, by instantiating the general definition.

Each of the states of $time(A,b)$ consists of As, a state of A, plus Ct, plus, for each class C of the partition, two times, $Ft(C)$ and $Lt(C)$. Each initial state of $time(A)$ consists of an initial state s of A, plus $Ct = 0$, plus values of $Ft(C)$ and $Lt(C)$ with the following property: if there is an action in C enabled in s, then $s.Ft(C) = b_\ell(C)$ and $s.Lt(C) = b_u(C)$. Otherwise, $s.Ft(C) = 0$ and $s.Lt(C) = \infty$.

If (π,t) is an action of $time(A)$, then $(s',(\pi,t),s)$ is a step of $time(A)$ exactly if the following conditions hold.

1. $(s'.As,\pi,s.As)$ is a step of A.
2. $s'.Ct \leq t = s.Ct$.

3. If $\pi \in C$, then

   (a) $s'.Ft(C) \leq t \leq s'.Lt(C)$.

   (b) if $s.AS \in enabled(A, C)$, then $s.Ft(C) = t + b_\ell(C)$ and $s.Lt(C) = t + b_u(C)$, and

   (c) if $s.AS \in disabled(A, C)$, then $s.Ft(C) = 0$ and $s.Lt(C) = \infty$.

4. For all classes D such that π is not in class D,

   (a) $t \leq s'.Lt(D)$,

   (b) if $s.As \in enabled(A, D)$ and $s'.As \in disabled(A, D)$ then $s.Ft(D) = t + b_\ell(D)$ and $s.Lt(D) = t + b_u(D)$,

   (c) if $s.As \in enabled(A, D)$ and $s'.As \in enabled(A, D)$ then $s.Ft(D) = s'.Ft(D)$ and $s.Lt(D) = s'.Lt(D)$, and

   (d) if $s.As \in disabled(A, D)$ then $s.Ft(D) = 0$ and $s.Lt(D) = \infty$.

In this special case, it is easy to check that for any class C of the partition, any reachable state s in which the $Lt(C)$ and $Ft(C)$ components have non-default values must have $s.As \in enabled(A, C)$. This definition is obtained from the general one by direct application of the definitions; the only condition that may appear to be slightly different is 4(b), where the general definition uses a min expression for the new value of $Lt(U)$. However, in the special case, any reachable state s' in which case 4(b) applies must have $s'.As \in disabled(A, D)$; therefore, the remark above implies that the first term in the min expression always has the value ∞, and so the min expression can be simplified as given.

It is possible to make stronger connections between the executions of $time(A, b)$ and the timed executions of (A, b) than those we have already described for the general case. To do this, we require one further definition.

Definition 3.1 *The* complete executions *of $time(A, b)$ are those executions α of $time(A, b)$ that satisfy one of the following conditions.*

1. *α is infinite and the time components of the actions in α are unbounded, or*

2. *α is finite and each locally controlled action of $time(A, b)$ that is enabled in $s_{end}(\alpha)$ is in a partition class C in $part(A)$ having $b_u(C) = \infty$.*

The complete schedules *and* complete behaviors *of $time(A)$ are defined to be the schedules and behaviors, respectively, of complete executions of $time(A)$.*

The following technical lemma says that the existence of *some* enabled action in $time(A, b)$ (as used in Condition 2. of the above definition) is equivalent to the existence of some enabled action in A.

Lemma 3.3 *Let (A,b) be a timed automaton. If α is a finite execution of $time(A,b)$, then α is complete exactly if for every class $C \in part(time(A,b))$,*

$$s_{end}(\alpha).As \in enabled(time(A,b),C) \Longrightarrow b_u(C) = \infty.$$

Proof: Denote $s = s_{end}(\alpha)$. It suffices to show, that a locally controlled action (π,t) of $time(A,b)$ is enabled in s, where $\pi \in C$ for some class C with $b_u(C) < \infty$, exactly if for some class C' with $b_u(C) < \infty$ an action $\phi \in C'$ is enabled in $s.As$.

Suppose a locally controlled action (π,t) of $time(A,b)$, where π is in a partition class C with $b_u(C) < \infty$, is enabled in s. Then Condition 1. of the definition of $time(A,b)$ immediately implies that π is enabled in $s.As$.

Conversely, suppose that a locally controlled action of A in a partition class C' with $b_u(C') < \infty$ is enabled in $s.As$; note that in this case the definition of $time(A,b)$ implies that $s.Lt(C') < \infty$. Let ϕ be a locally controlled action of A that is enabled in $s.As$ and that is in a class C'' having the minimal value of $s.Lt(C)$ for any class C. We have that $s.Lt(C'') \leq s.Lt(C') < \infty$; furthermore, it follows from the definition of $time(A,b)$ that for any class C, if $b_u(C) = \infty$ then $Lt(C)$ components are always ∞. Thus it follows that $b_u(C'') < \infty$. From Conditions 3(a) and 4(a) in the definition of $time(A,b)$ it follows that (ϕ,t) is enabled in s, where $t = qs.Lt(C'')$. ■

Now we relate the timed executions of a timed automaton (A,b) to the complete executions of the corresponding I/O automaton $time(A,b)$.

Theorem 3.4 *1. If α' is a timed execution of (A,b), then there exists a complete execution α of $time(A,b)$ such that $\alpha' = project(\alpha)$.*

2. If α is a complete execution of $time(A,b)$, then $project(\alpha)$ is a timed execution of (A,b).

Proof: 1. Suppose α' is a timed execution of (A,b). If α' is infinite then Lemma 3.2 implies that there is an infinite execution α of $time(A,b)$ in which the time components are unbounded, such that $\alpha' = project(\alpha)$. Then α is a complete execution of $time(A,b)$, as needed. So assume that α' is finite. Then Lemma 3.1 implies that there is a finite execution α of $time(A,b)$ such that $\alpha' = project(\alpha)$. By Lemma 2.2, every locally controlled action of A that is enabled in the final state of α' is in a partition class with upper bound equal to ∞. Then every locally controlled action of $time(A,b)$ that is enabled in the final state of α is in a partition class with upper bound equal to ∞, by Condition 1. of the definition of $time(A,b)$. Therefore, α is a complete execution, as needed.

2. Suppose that α is a complete execution of $time(A,b)$. If α is infinite then the claim follows from Lemma 3.2, so suppose that α is finite. Then Lemma 3.1 implies that $project(\alpha)$ is a timed semi-execution of (A,b); it remains to show the liveness part of the upper bound condition of Definition 2.2. Lemma 3.3 implies that every locally controlled action of A

that is enabled in $s_{end}(\alpha).As$ is in a partition class with upper bound equal to ∞. But this state is the same as the final state of $project(\alpha)$. Therefore, $project(\alpha)$ satisfies the liveness part of the upper bound vacuously.

■

Corollary 3.5 *The set of timed behaviors of* (A,b) *is the same as the set of complete behaviors of* $time(A,b)$.

This theorem implies that properties of timed behaviors of a timed automaton (A,b) can be proved by proving them about the set of complete behaviors of the corresponding I/O automaton $time(A,b)$. The latter task is more amenable to treatment using assertional techniques.

In each of our examples in this paper, we will apply the $time(A,b)$ construction to a timed automaton A modeling the entire system.

4 Resource Manager

Now we present our first example, a simple resource-granting system adapted from an algorithm in [AtL89]. The system consists of two components, a *clock* and a *manager*. The clock ticks at an approximately-predictable rate, and the manager counts ticks in order to decide when to grant a resource. We wish to analyze the time until the first grant, and the time between each successive pair of grants.

We describe the algorithm and its timing assumptions as a timed automaton (A,b). The required timing behavior is presented as a set of timing conditions $\mathcal{U}$; we prove that the algorithm satisfies the requirements by demonstrating a strong possibilities mapping from $time(A,b)$ to $time(A,\mathcal{U})$.

4.1 The Algorithm

The algorithm consists of two components, a *clock* and a *manager*. The *clock* has only one action, the output *TICK*, which is always enabled, and has no effect on the clock's state. It can be described as the particular one-state automaton with the following steps.[5]

TICK
Precondition:
true

[5]In the notation we use for automata, a separate description is given for the steps involving each action. Instead of listing the steps, we provide a "precondition" which describes the set of states in which the action is enabled, and an "effect" which describes the changes caused by the action. Input actions do not have a precondition, because they are always enabled.

Effect:
none

The boundmap associates the interval $[c_1, c_2]$, where $0 < c_1 \leq c_2 < \infty$, with the single class, $\{TICK\}$, of the partition. For convenience, we overload the notation and designate this singleton class as *TICK* also. This means that successive *TICK* events occur with intervening times in the given interval.

The manager has one input action, *TICK*, one output action, *GRANT* and one internal action, *ELSE*. The manager waits a particular number $k > 0$ of clock ticks before issuing each *GRANT*, counting from the beginning or from the last preceding *GRANT*. The manager's state has one component: TIMER, holding an integer, initially k.

The manager's algorithm is as follows:

TICK
Effect:
TIMER := TIMER -1

GRANT
Precondition:
TIMER ≤ 0
Effect:
TIMER := k

ELSE
Precondition:
TIMER > 0
Effect:
none

Notice that *ELSE* is enabled exactly when *GRANT* is not enabled. The effect of including the *ELSE* action is to ensure that the automaton continues taking steps at its "own pace", at approximately regular intervals.

Thus, in the situation we are modeling, when the *GRANT* action's precondition becomes satisfied, the action does not occur instantly – the action waits until the automaton's next local step occurs.[6]

The partition groups the *GRANT* and *ELSE* actions into a single equivalence class *LOCAL*, with which the boundmap associates the interval $[0, l]$, where $0 \leq l < \infty$. We assume that

[6] An alternative situation is one in which the manager is interrupt-driven, that is, whenever the precondition of a *GRANT* becomes true, the *GRANT* occurs shortly thereafter. This situation could be modeled by omitting the *ELSE* action. The two automata have slightly different timing properties. In this paper, we only consider the first assumption.

$c_1 > l$.[7] Fix A to be the I/O automaton which is the composition of the clock and manager, with the $TICK$ output action converted to an internal action; thus, the only external action of A is the output action $GRANT$. Also, let b be the boundmap described above. We wish to show that all the timed behaviors of (A, b) satisfy certain upper and lower bounds on the time up to the first $GRANT$ and the time between consecutive pairs of $GRANT$ events.

Note that our resource manager is much simpler than the usual examples; in particular, there is no $REQUEST$ input action that triggers the $GRANT$ output. We do not think that such added structure would add much to the conceptual difficulty of the example or expose any interesting property of the methodology we suggest here; however, it would make the analysis somewhat longer.

We begin our analysis by stating some invariant properties of the algorithm. In order to do this, we need timing information to be included in the state, so we consider the automaton $time(A, b)$, constructed as described in Section 3.2. Notice that in this case, the automaton $time(A, b)$ has the following components, As, Ct, $Ft(TICK)$, $Lt(TICK)$, $Ft(LOCAL)$, and $Ft(LOCAL)$.

The next lemma states invariant properties of the automaton $time(A, b)$. Notice that the second property involves the time components of the state. The proof of this lemma is fairly technical and appears in full detail in Appendix A.

Lemma 4.1 *The following are true about any reachable state s of $time(A, b)$.*

1. $s.TIMER \geq 0$.
2. *If* $s.TIMER = 0$ *then* $s.Ft(TICK) \geq s.Lt(LOCAL) + c_1 - l$.

We close this subsection with a proof of a basic property of $time(A, b)$ (for this fixed (A, b)).

Lemma 4.2 *All complete executions (and therefore all complete schedules) of $time(A, b)$ are infinite.*

Proof: Suppose by way of contradiction that α is a finite complete execution of $time(A, b)$. Then by Lemma 3.3, every locally controlled action of A that is enabled in $s_{end}(\alpha).As$ is in a partition class with upper bound equal to ∞. But $TICK$ is always enabled and $b_u(TICK) = c_2 < \infty$, which is a contradiction. ■

This lemma tells us that for this example we do not have to worry about the case where executions are finite – we can assume that we have infinite executions in which (because of the definition of completeness) the timing component is unbounded.

[7] Again, a different assumption would change the timing analysis.

4.2 The Requirements Automaton

We wish to show the following, for any timed behavior β of (A, b):

1. There are infinitely many $GRANT$ events in β.
2. If t is the time of the first $GRANT$ event in β, then $k \cdot c_1 \leq t \leq k \cdot c_2 + l$.
3. If t_1 and t_2 are the times of any two consecutive $GRANT$ events in β, then

$$k \cdot c_1 - l \leq t_2 - t_1 \leq k \cdot c_2 + l.$$

We let $\mathbf{P}$ denote the set of sequences of $(action, time)$ pairs satisfying the above three conditions.

By the earlier characterization, Corollary 3.5, it suffices to show that all complete behaviors of $time(A, b)$ are in $\mathbf{P}$.

We will specify $\mathbf{P}$ in terms of another I/O automaton, called the *requirements automaton*. We define two timing conditions, G_1 for the time until the initial $GRANT$ event and G_2 for the time between successive $GRANT$ events. The requirements automaton B is defined to be $time(A, \{G_1, G_2\})$.

We now define the conditions. The first condition, G_1, is $(T_{start}(G_1), \emptyset) \overset{b(G_1)}{\rightsquigarrow} (\Pi(G_1), \emptyset)$, where

- $T_{start}(G_1)$ is the (singleton) set of start states of A,
- $b_\ell(G_1) = k \cdot c_1$ and $b_u(G_1) = k \cdot c_2 + l$, and
- $\Pi(G_1) = \{GRANT\}$.

The second condition, G_2, is $(\emptyset, T_{step}(G_2)) \overset{b(G_2)}{\rightsquigarrow} (\Pi(G_2), \emptyset)$, where

- $T_{step}(G_2) = \{(s', \pi, s) \in steps(A) : \pi = GRANT\}$,
- $b_\ell(G_2) = k \cdot c_1 - l$ and $b_u(G_2) = k \cdot c_2 + l$, and
- $\Pi(G_2) = \{GRANT\}$.

Now we prove a lemma that relates the behaviors of B to the condition $\mathbf{P}$. Note that the behaviors of B and the sequences in $\mathbf{P}$ both consist of elements that are pairs, an action of A together with a time.

Lemma 4.3 *Let β be an infinite schedule of B in which the time component is unbounded. Then $beh(\beta)$ is in $\mathbf{P}$.*

Proof: First notice that if α is a timed execution of $(A, \{G_1, G_2\})$ then $beh(\alpha)$ is in $\mathbf{P}$.

Let β be an infinite schedule of $B = time(A, \{G_1, G_2\})$ in which the time component is unbounded; then $\beta = sched(\alpha)$, where α is an infinite execution of $time(A, \{G_1, G_2\})$ in which the time component is unbounded.

Then Lemma 3.2 implies that $\alpha' = project(\alpha)$ is a timed execution of $(A, \{G_1, G_2\})$. Therefore by the remark above, $beh(\alpha') \in \mathbf{P}$. But $beh(\beta) = beh(\alpha) = beh(\alpha')$, so $beh(\beta) \in \mathbf{P}$, as needed. ■

Thus, it suffices to show that every complete behavior of $time(A, b)$ is of the form $beh(\beta)$, where β is an infinite schedule of B in which the time component is unbounded. Now, every complete behavior of $time(A, b)$ is of the form $beh(\beta)$, where β is a complete schedule of $time(A, b)$. But every complete schedule of $time(A, b)$ is infinite; moreover, the definition of completeness for infinite executions implies that the time component of β is unbounded. It suffices to show that β is a schedule of B.

Thus, it remains to show that every schedule of $time(A, b)$ is also a schedule of B. But this is precisely the kind of task that a strong possibilities mapping can be used to carry out; the complete formal proof appears in the next section.

4.3 The Mapping

In this section, we present a strong possibilities mapping from $time(A, b)$ to B, thereby showing that all schedules of $time(A, b)$ are also schedules of B. This fact is then used to prove Theorem 4.5, which says that all timed behaviors of (A, b) are in $\mathbf{P}$.

We define a mapping f so that a state u of B is in the image set $f(s)$ exactly if the following conditions hold.

1. If $s.TIMER > 0$ then
   (a) $min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(TICK) + (s.TIMER - 1)c_2 + l$, and
   (b) $max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ft(TICK) + (s.TIMER - 1)c_1$.
2. If $s.TIMER = 0$ then
   (a) $min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(LOCAL)$, and
   (b) $max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ct$.

The inequalities should be interpreted as giving explicit upper and lower bounds for the time of the next $GRANT$ event, in terms of the values of the variables in the state of $time(A, b)$. Intuitively, the right-hand side of the inequality describes *how* the bounds will be satisfied; for example, in the case of inequality 1(a), a $TICK$ event must happen within time $Lt(TICK)$.

and then after TIMER − 1 additional ticks, each happening after at most c_2 time, TIMER will become 0, thus enabling the $GRANT$, which will happen within at most time l.

If we think of the value of $min(Lt(G_1), Lt(G_2))$ as indicating an upper bound on the time when a $GRANT$ will next occur, then it should not be surprising that any sufficiently large number (with respect to the values of the variables in the state of $time(A)$) could be used as the value of this minimum. This just indicates that any such value could be proved to be an upper bound. Similarly, any sufficiently small number could be used as the value for $max(Ft(G_1), Ft(G_2))$, a lower bound on the time when a $GRANT$ event will next occur.

Thus, the inequalities comprising the strong possibilities mapping express the fact that *any* sufficiently large number (with respect to the values of the variables in the state of $time(A, b)$) should be provable as an upper bound for the time for the next $GRANT$, and any sufficiently small number should be provable as a lower bound.[8]

The given mapping is obviously multivalued, because it is described in terms of inequalities. It seems possible to use a single-valued mapping for this example by complicating the definition of the requirements automaton; however, since the requirements automaton is serving as the problem specification, that does not seem like a good idea. More discussion of the issue of multivalued vs. single-valued mappings appears in [Ly89].

Although (we think that) the correspondence between $time(A, b)$ and B described by f is easy to understand, verifying formally that f is indeed a strong possibilities mapping is a fairly long and mechanical process. The complete proof appears in Appendix A.

Lemma 4.4 *The mapping f is a strong possibilities mapping.*

Now we can put the pieces together.

Theorem 4.5 *All timed behaviors of (A, b) are in* **P**.

Proof: Let γ be a timed behavior of (A, b). Then by Corollary 3.5, γ is a complete behavior of $time(A, b)$. Let β be a complete schedule of $time(A, b)$ such that $\gamma = beh(\beta)$. By Lemma 4.2, β is infinite, and by the definition of completeness for infinite executions, the time components of β are unbounded. Since, by Lemma 4.4, there is a strong possibilities mapping from $time(A, b)$ to B, Lemma 2.1 implies that β is also a schedule of B. Since β is an infinite schedule of B in which the time components are unbounded, Lemma 4.3 implies that $beh(\beta) = \gamma$ is in **P**. ■

[8] Note that if we simply replaced the inequalities with equations, the resulting mapping would not be a strong possibilities mapping. For example, suppose that a clock tick occurs within less than the maximum c_2. Then the right-hand side expression in 1(a) would evaluate after the step to an earlier time than before the step. On the other hand, the corresponding step in the requirements automaton would not change the value of $Ltime(GRANT)$, the correspondence thus would not be preserved.

As we promised in the Introduction, our mapping proof has here yielded all the timing properties we require, including both safety and liveness properties. The mapping immediately yields the safety properties. (Recall that the safety properties are the lower bounds, as well as the upper bounds that assert that time cannot elapse without a certain event having occurred.) But when these safety properties are combined with the property that all complete executions are infinite and our assumption that the time in infinite timed executions is unbounded (so that time increases without bound), they also imply that the events in question must eventually occur.

5 Signal Relay

Now we present our second example, a simple signal relay. The system is a composition of a collection of $n + 1$ processes, $P_0, \ldots, P_n$, organized as a *line*. P_0 generates $SIGNAL_0$ (once), and the intermediate processes relay it, so that P_n eventually generates $SIGNAL_n$. We wish to analyze the total delay a signal incurs, as a function of its delay at each of the relaying processes.

Again, we describe the algorithm and its timing assumptions as a timed automaton (A, b), and the required timing behavior as a set of timing conditions $\mathcal{U}$. This time, however, we do not simply present a direct mapping from $time(A,b)$ to $time(A,\mathcal{U})$ (although we could have). Rather, we use a sequence of intermediate automata, exhibiting strong possibilities mappings between each consecutive pair of automata in the sequence. The style of the reasoning involved corresponds closely to that of a proof based on recurrence inequalities. (In fact, this example was inspired by the recurrence-inequality proof sketch in [LG89] for the tournament mutual exclusion algorithm of [PF77]).

5.1 The Algorithm

The algorithm consists of $n + 1$ automata, $P_0, \ldots, P_n$, where $n \geq 1$. P_0 has one action, the output $SIGNAL_0$. The state of P_0 consists of one component, FLAG, a Boolean value, initially *true*.

P_0's algorithm is as follows:

$SIGNAL_0$
Precondition:
    FLAG = *true*
Effect:
    FLAG := *false*

The boundmap associates the interval $[0, \infty]$ with the single class, $\{SIGNAL_0\}$, of the partition. As before, we also designate this class as $SIGNAL_0$; we use similar notational conventions for the remaining singleton classes in the paper.

Each automaton P_i, $1 \leq i \leq n$, has an input action $SIGNAL_{i-1}$ and an output action $SIGNAL_i$. Each automaton state contains the single component FLAG, holding a Boolean value, initially *false*.

The algorithm for P_i is:

$SIGNAL_{i-1}$
Effect:
    FLAG := *true*

$SIGNAL_i$
Precondition:
    FLAG = *true*
Effect:
    FLAG := *false*

The boundmap associates the interval $[d_1, d_2]$, where $0 \leq d_1 \leq d_2 < \infty$, with the single class, $SIGNAL_i$, of the partition for P_i.

Now we fix A to be the timed automaton which is the composition of all the P_i's, with all actions except $SIGNAL_0$ and $SIGNAL_n$ made internal, and b to be the boundmap described above. We will prove that if a $SIGNAL_0$ occurs, then the difference between the time it occurs and the time at which a later $SIGNAL_n$ occurs is at least $n \cdot d_1$ and at most $n \cdot d_2$.

We first state the following simple invariant about A. The proof is by a simple induction.

Lemma 5.1 *In any reachable state s of A, if $SIGNAL_i$ is enabled in s, then for all $j \neq i$, $0 \leq j \leq n$, $SIGNAL_j$ is not enabled in s.*

Now recall that in the previous example, we had the property, expressed in Lemma 4.2, that all complete executions were infinite. This property was very useful in establishing the liveness part of the upper bound result, in Theorem 4.5. Unfortunately, in the present example we do not have this property: the automaton $time(A, b)$ has complete executions that are finite. (In fact, all of its executions are finite.) Since it seems much more straightforward to use our proof method when all complete executions are infinite, we find it useful to define a variant (A', b') of the timed automaton (A, b), which augments A with a "dummy" component that always has locally-controlled actions enabled. All of the complete executions of (A', b') will be infinite, and the executions of $time(A, b)$ and $time(A', b')$ are very closely related. Thus, we will be able to reason about (A', b') and obtain consequences for the original timed automaton (A, b).

More formally, we augment system A with a single new component called the *dummy*. The *dummy* has a single action, an output $NULL$ (which is not shared by any of the other components). It has only one state, in which $NULL$ is enabled. The boundmap associates any interval $[n_1, n_2]$ such that $0 \leq n_1 \leq n_2 < \infty$ with the new singleton partition class, $NULL$. Let A' be the automaton composed of all the P_i and the *dummy*, with all actions except $SIGNAL_0$ and $SIGNAL_n$ made internal. Also, let b' be the boundmap that is identical to b except for the addition of the new interval $[n_1, n_2]$ for the new partition class, $NULL$.

Lemma 5.2 *All complete executions (and therefore all complete schedules) of time(A', b') are infinite.*

Proof: Suppose by way of contradiction that α is a finite complete execution of $time(A', b')$. Then by Lemma 3.3, every locally controlled action of A that is enabled in $s_{end}(\alpha)$.As is in a partition class with upper bound equal to ∞. But $NULL$ is always enabled and $b_u(NULL) = n_2 < \infty$, which is a contradiction. ■

Lemma 5.3 *Let β be a complete schedule of time(A, b). Then there exists a complete schedule β' of time(A', b') such that β can be obtained from β' by removing $NULL$ events.*

Proof: Let α be a complete execution of $time(A, b)$ such that $\beta = sched(\alpha)$. We construct a complete execution α' of $time(A', b')$ by inserting an infinite sequence of $NULL$ events into α, at times that are separated by some fixed nonzero time increment in the interval $[n_1, n_2]$. (The state component must also be augmented appropriately.) We then let $\beta' = sched(\alpha')$. We leave to the reader the verification that β' has the required properties. ■

Since the $NULL$ action is internal we have the following corollary.

Corollary 5.4 *Let β be a complete behavior of time(A, b). Then β is also a complete behavior of time(A', b').*

5.2 The Requirements Automaton

We wish to show the following, for any timed behavior β of (A, b):

1. If $SIGNAL_0$ event occurs in β, then a single later $SIGNAL_n$ event occurs in β.
2. If t_1 is the time of a $SIGNAL_0$ event and t_2 is the time of the corresponding $SIGNAL_n$ event then:
$$n \cdot d_1 \leq t_2 - t_1 \leq n \cdot d_2.$$

We let **Q** denote the set of sequences of $(action, time)$ pairs satisfying the above two conditions.

By Corollary 3.5, it suffices to show that all complete behaviors of $time(A, b)$ are in **Q**. We will show that all complete behaviors of $time(A', b')$ are in **Q**. It will then follow by Corollary 5.4 that all complete behaviors of $time(A, b)$ are in **Q**.

We now specify **Q** in terms of a requirements automaton. Towards this end, we define the following timing condition, $U_{0,n} = (\emptyset, T_{0,n}) \overset{b_{0,n}}{\rightsquigarrow} (\Pi_{0,n}, \emptyset)$, where

- $T_{0,n} = \{(s', \pi, s) \in steps(A') : \pi = SIGNAL_0\}$,
- $b_{0,n} = [n \cdot d_1, n \cdot d_2]$ and
- $\Pi_{0,n} = \{SIGNAL_n\}$.

The requirements automaton B is $time(A', \{U_{0,n}\})$.

Lemma 5.5 *Let β be an infinite schedule of B in which the time component is unbounded. Then $beh(\beta) \in \mathbf{Q}$.*

Proof: First notice that if α is a timed execution of $(A', \{U_{0,n}\})$ then $beh(\alpha)$ is in **Q**.

Let β be an infinite schedule of $B = time(A', \{U_{0,n}\})$ in which the time component is unbounded; then $\beta = sched(\alpha)$, where α is an infinite execution of $time(A', \{U_{0,n}\})$ in which the time component is unbounded. Then Lemma 3.2 implies that $\alpha' = project(\alpha)$ is a timed execution of $(A', \{U_{0,n}\})$. Therefore by the remark above, $beh(\alpha') \in \mathbf{Q}$. But $beh(\beta) = beh(\alpha) = beh(\alpha')$, so $beh(\beta) \in \mathbf{Q}$, as needed. ■

So again it suffices to show that every complete behavior of $time(A', b')$ is of the form $beh(\beta)$, where β is an infinite schedule of B in which the time component is unbounded. Now, every complete behavior of $time(A', b')$ is of the form $beh(\beta)$, where β is a complete schedule of $time(A', b')$. But every complete schedule of $time(A', b')$ is infinite, and the definition of completeness for infinite executions implies that the time component of β is unbounded. It remains to show that β is a schedule of B.

Thus, it is enough to show that every schedule of $time(A', b')$ is also a schedule of B; this is done in the following subsections.

5.3 The Intermediate Requirements Automata

One way of proceeding would be to exhibit a strong possibilities mapping directly from $time(A', b')$ to B, following the pattern of the first example. However, an alternative and attractive strategy might be based on the recursive structure of the line of processes. For instance, one might give a recursive analysis of the time between any $SIGNAL_k$, $0 \leq k \leq n-2$

and $SIGNAL_n$ in terms of the time between $SIGNAL_{k+1}$ and $SIGNAL_n$. Thus, the analysis would be based on recurrence inequalities. Several examples of such recurrence inequality analyses (for upper bounds only) appear in [LG89]; the analysis of the Peterson-Fischer ([PF77]) tournament algorithm in [LG89, p. 26–30] is a particularly good example of this proof style.

Recurrence inequality proofs, however, have an "operational" style that is very different from the assertional style we are describing here. We would like to be able to utilize the power of the recurrence analysis within our assertional framework. In order to do this, instead of proceeding to show directly that every schedule of $time(A', b')$ is a schedule of B by a strong possibilities mapping, we proceed using a hierarchy of intermediate requirements automata. Each intermediate requirements automaton, B_k, includes the same timing conditions as are given by the boundmap b', for partition classes $SIGNAL_0, ...SIGNAL_k$, plus a new timing condition that provides bounds on the time between $SIGNAL_k$ and a subsequent $SIGNAL_n$. The recursive argument described above, expressing the time between $SIGNAL_k$ and $SIGNAL_n$ in terms of the time between $SIGNAL_{k+1}$ and $SIGNAL_n$, is then captured formally by a strong possibilities mapping from B_k to B_{k+1}.

In this subsection, we define the intermediate automata.

First, for every k, $0 \leq k \leq n-1$, we define a timing condition stating that the time between $SIGNAL_k$ and $SIGNAL_n$ (if $SIGNAL_k$ occurs) is in the interval $[(n-k)d_1, (n-k)d_2]$. (In particular, the condition will imply that each $SIGNAL_k$ is actually followed by a corresponding $SIGNAL_n$). When $k = n-1$, this condition will be the same as the timing condition assigned by the boundmap b' to the class containing $SIGNAL_n$. On the other hand, when $k = 0$, this condition is the same as the condition $U_{0,n}$ previously defined, i.e., the timing condition we wish to prove.

Formally, for any $0 \leq k \leq n-1$,[9] we define the following timing condition, $U_{k,n} = (\emptyset, T_{k,n}) \overset{b_{k,n}}{\rightsquigarrow} (\Pi_{k,n}, \emptyset)$, where

- $T_{k,n} = \{(s', \pi, s) \in steps(A') : \pi = SIGNAL_k\}$,
- $b_{k,n} = [(n-k) \cdot d_1, (n-k) \cdot d_2]$, and
- $\Pi_{k,n} = \{SIGNAL_n\}$.

For any $k, 0 \leq k \leq n-1$, let $\mathcal{U}_k$ be the set of timing conditions that includes $U_{k,n}$ and the conditions assigned by boundmap b' to the partition classes $SIGNAL_0, ..., SIGNAL_k$ and $NULL$. Let B_k denote the I/O automaton $time(A', \mathcal{U}_k)$.

In the next subsection, we will show the existence of a strong possibilities mapping from B_k to B_{k-1}, for every $k, 1 \leq k \leq n-1$. This implies that there is a strong possibilities mapping from B_{n-1} to B_0. Moreover, there is a trivial strong possibilities mapping from B_0 to the requirements automaton B (which just ignores the timing conditions associated

[9] The redefinition of $U_{0,n}$ is consistent with the prior definition.

by b' with the partition classes $SIGNAL_0$ and $NULL$). Similarly, there is a trivial strong possibilities mapping from $time(A', b')$ to B_{n-1} (which simply renames the state components associated with $SIGNAL_n$). Therefore, this mapping proof will imply the existence of a strong possibilities mapping from $time(A', b')$ to B.

5.4 The Mapping

In this subsection, we fix a particular value of $k, 1 \leq k \leq n-1$, and show the existence of a strong possibilities mapping, f_k, from B_k to B_{k-1}.

Recall that the timing conditions included in B_k are those for $U_{k,n}$, $SIGNAL_0, ..., SIGNAL_k$ and $NULL$, while those included in B_{k-1} are those for $U_{k-1,n}$, $SIGNAL_0, ..., SIGNAL_{k-1}$ and $NULL$, For the sake of convenience we denote by $Ft(k,n)$ (respectively, $Lt(k,n)$) the Ft (respectively, Lt) component of the state of B_k that is associated with $U_{k,n}$. Also, as we did in our construction of $time(A,b)$, we denote by $Ft(C)$ (respectively, $Lt(C)$) the Ft (respectively, Lt) components that are associated by the boundmap b' with each partition class C. We also use the notation $FLAG_i$, $0 \leq i \leq n$, to denote the $FLAG$ component of P_i.

Now we define f_k so that a state $u \in states(B_{k-1})$ is in the image set $f_k(s)$, for $s \in states(B_k)$, exactly if the following hold.

$$u.Lt(k-1,n) \geq \begin{cases} s.Lt(k,n) & \text{if } s.\text{FLAG}_i = true \\ & \text{for some } i, k+1 \leq i \leq n \\ s.Lt(SIGNAL_k) + (n-k)d_2 & \text{if } s.\text{FLAG}_k = true \\ \infty & \text{otherwise,} \end{cases}$$

and

$$u.Ft(k-1,n) \leq \begin{cases} s.Ft(k,n) & \text{if } s.\text{FLAG}_i = true \\ & \text{for some } i, k+1 \leq i \leq n \\ s.Ft(SIGNAL_k) + (n-k)d_1 & \text{if } s.\text{FLAG}_k = true \\ 0 & \text{otherwise.} \end{cases}$$

and every other component of state u of B_{k-1} is equal to the corresponding component of the state s; notice that by Lemma 5.1 if $\text{FLAG}_k = true$ then $\text{FLAG}_i = false$ for all $i \neq k$, $0 \leq i \neq n$, thus the mapping is well defined.

Intuitively, the inequalities give upper and lower bounds for the time of the next $SIGNAL_n$ event, in terms of the values of the variables in the state of $time(A', b')$. For example, in the case of the upper bound, if the signal has already propagated past process P_k, then within the time that is stored in $s.Lt(k,n)$, a $SIGNAL_n$ event must occur (because the component $s.Lt(k,n)$ keeps track of the latest time at which a $SIGNAL_n$ event must occur, once a $SIGNAL_k$ event has occurred). If the signal has only gotten as far as process P_k, however, then $s.Lt(k,n)$ will not contain any useful information, so an alternative bound is used. In this case, within time $s.Lt(SIGNAL_k)$, a $SIGNAL_k$ event must occur, and then after $(n-k)$ additional signal

propagation steps, each taking at most time d_2, a $SIGNAL_n$ event must occur. The lower bound has a similar meaning.

The proof of the following lemma is a straightforward case analysis and it appears in Appendix A.

Lemma 5.6 *If $1 \leq k \leq n-1$ then the mapping f_k is a strong possibilities mapping from B_k to B_{k-1}.*

By considering the composition $f_1 \circ \cdots \circ f_{n-1}$ and the trivial mappings from B_0 to B and from $time(A', b')$ to B_{n-1}, we obtain the following corollary.

Corollary 5.7 *There exists a strong possibilities mapping from $time(A', b')$ to B.*

Now we can put the pieces together.

Theorem 5.8 *All timed behaviors of (A, b) are in* **Q**.

Proof: Let γ be a timed behavior of (A, b). Then by Corollary 3.5, γ is a complete behavior of $time(A, b)$. Thus, Corollary 5.4 implies that γ is a complete behavior of $time(A', b')$. Let β be a complete schedule of $time(A', b')$ such that $\gamma = beh(\beta)$. By Lemma 5.2, β is infinite, and by the definition of completeness for infinite executions, the time components of β are unbounded. Since by Corollary 5.7 there is a strong possibilities mapping from $time(A', b')$ to B, Lemma 2.1 implies that β is also a schedule of B. Since β is an infinite schedule of B in which the time components are unbounded, Lemma 5.5 implies that $beh(\beta) = \gamma$ is in **Q**. ■

6 Conclusions and Further Work

In this paper we have described a way to carry out assertional proofs for timing properties. We have shown how to specify an algorithm and its timing assumptions, as well as its performance requirements, in terms of timed automata and timing conditions. We have shown how to convert such specifications into ordinary (not timed) I/O automata by building predictive timing information into the automaton states. Then the goal of proving timing conditions can often be met by demonstrating the existence of a strong possibilities mapping from the automaton corresponding to the algorithm (with its timing assumptions) to the automaton corresponding to the performance requirements.

We have presented two examples of this method. The first is the analysis of the rate at which a simple resource manager system issues grants; the second is the analysis of the propagation delay of a signal along a line of relay processes. The second example also illustrates how our method can be applied hierarchically, in a way that corresponds to proofs using recurrences.

A good technique for proving timing properties of timing-dependent or asynchronous systems should be rigorous, simple and general. Our technique is certainly rigorous, and we think it is also quite simple. Prior work on proving timing properties has usually had an operational style much like that of liveness proofs, where time bounds are obtained by bounding how long it takes for intermediate milestones to occur. (See [LG89] for several examples.) In contrast, the method presented in this paper has an assertional style. Such a style seems to us to lead to proofs that are somewhat simpler; they are straightforward to generate (although they may involve analyzing a large number of cases), and are easier to check – in fact, proofs of the sort we have given in this paper ought to be machine-checkable with current proof technology.

As for generality, it is not yet clear to us how generally applicable this method will be. It is quite likely that the specific $time(A,\mathcal{U})$ construction we use will not be general enough to express all interesting examples of performance requirements. For example, one might want to consider performance requirements that specify that a resource manager is supposed to respond to requests as long as they do not arrive too far apart in time (see the "cement mixer" example in [FG89]). For another example, one might want to consider a specification that says that one event π triggers two later events, ϕ and ψ, with ϕ occurring within a certain interval of time after π and ψ occurring within a certain interval of time after ϕ. Both of these examples illustrate more complicated requirements than can be expressed directly as timing conditions. It may be possible to force such examples to fit into our definitions by adding auxiliary variables or actions; alternatively, it may be necessary or desirable to generalize the $time(A,\mathcal{U})$ construction to allow more general kinds of timing conditions. If the $time(A,\mathcal{U})$ construction is generalized, then we would hope that many of the same ideas, e.g., the incorporation of predictive timing information into the state and the use of mappings that take the form of inequalities, will still be useful. Even if the $time(A,\mathcal{U})$ construction is generalized, we wonder whether there is a single generalization that will cover all interesting examples. We leave all of this as a subject for future work.

Even for the specific case of mappings between automata of the form $time(A,\mathcal{U})$, we do not yet know if our proof method using inequalities is complete. That is, if every schedule of $time(A,\mathcal{U}_1)$ is also a schedule of $time(A,\mathcal{U}_2)$ then is there necessarily a strong possibilities mapping (in the form of inequalities) from $time(A,\mathcal{U}_1)$ to $time(A,\mathcal{U}_2)$? Such completeness results for the usage of refinement mappings to prove properties of non timing-based algorithms appear in [AbL88] and [M89].

It remains to apply this technique to other, more complex examples than the ones in this paper. One particularly good example to try is the full tournament mutual exclusion algorithm from [PF77]. Its prior analysis using recurrences suggests that it may be a good candidate for hierarchical proof as in our second example. This is an example of an asynchronous algorithm; good sources for timing-dependent algorithms to analyze are the areas of real-time computing and communication.

We have already seen how our method can express ideas previously expressed using recurrences. It remains to see how our technique combines with other methods for time analysis such as methods based on bounded temporal logic (e.g., [BH81]). Also, it remains to see how

proofs using our techniques can be applied in a modular way for the verification of timing properties of large and complex timing-based systems.

Acknowledgements.

We would like to thank Steve Ponzio for his helpful comments on earlier versions of this paper.

References

[AbL88] M. Abadi and L. Lamport, "The Existence of Refinement Mappings," DEC SRC Research Report 29, August 1988.

[AtL89] H. Attiya and N. Lynch, "Time Bounds for Real-Time Process Control in the Presence of Timing Uncertainty," to appear in *proc. 10th Real-Time Systems Symposium*, December 1989. Expanded version available as Technical Report MIT/LCS/TR-403, Laboratory for Computer Science, MIT, July 1989.

[BH81] A. Bernstein and P. Harter, Jr. "Proving Real-Time Properties of Programs with Temporal Logic," in *Proc. 8th Symp. on Operating System Principles*, Operating Systems Review, Vol. 15, No. 5 (December 1981), pp. 1–11.

[FG89] M. W. Franklin and A. Gabrielian, "A Transformational Method for Verifying Safety Properties in Real-Time Systems," in *Proc. 10th IEEE Real-Time Systems Symp.*, December 1989, to appear. Also Technical Report 89-12, Tomson-CSF, Inc., July 1989.

[GF88] A. Gabrielian and M. W. Franklin, "State-Based Specification of Complex Real-Time Systems," in *Proc. 9th IEEE Real-Time Systems Symp.*, 1988, pp. 2–11.

[H81] V. H. Hasse, "Real-time behavior of programs," *IEEE Transactions on Software Engineering*, Vol. SE-7, No. 5 (September 1981), pp. 494–501.

[Ho87] J. Hooman, *A Compositional Proof Theory for Real-Time Distributed Message Passing*, TR. 4-1-1(1), Department of Mathematics and Computer Science, Eindhoven University of technology, March 1987.

[JM87] F. Jahanian and A. Mok, "A Graph-Theoretic Approach for Timing Analysis and Its Implementation," *IEEE Transactions on Computers*, Vol. C-36, No. 8 (August 1987), pp. 961–975.

[JS88] F. Jahanian and D. A. Stuart, "A Method for Verifying Properties of Modechart Specifications," in *Proc. 9th IEEE Real-Time Systems Symp.*, 1988, pp. 12–21.

[KVR83] R. Koymans, J. Vytopil and W. P. deRoever, "Real-Time Programming and Asynchronous Message Passing," in *Proc. 2nd ACM Symp. on Principles of Distributed Computing*, 1983, pp. 187–197.

[La83] L. Lamport, "Specifying Concurrent Program Modules," *ACM Trans. on Programming Languages and Systems*, Vol. 5, No. 2 (April 1983), pp. 190–222.

[H89] H. R. Lewis, "Finite-State Analysis of Asynchronous Circuits with Bounded Temporal Uncertainty," Technical Report TR-15-89, Aiken Computation Laboratory, Harvard University.

[Ly86] N. Lynch, "Concurrency Control for Resilient Nested transactions," *Advances in Computing Research*, Vol. 3, 1986, pp. 335–373.

[Ly88] N. Lynch, "Modelling Real-Time Systems," in *Foundations of Real-Time Computing Research Initiative*, ONR Kickoff Workshop, November 1988, pp. 1-16.

[Ly89] N. Lynch, "Multivalued Possibilities Mappings," REX Workshop, May 1989.

[LG89] N. Lynch and K. Goldman, *Lecture notes for 6.852*. MIT/LCS/RSS-5, Laboratory for Computer Science, MIT, 1989.

[LT87] N. Lynch and M. Tuttle, "Hierarchical Correctness Proofs for Distributed Algorithms," in *Proc. 7th ACM symp. on Principles of Distributed Computing*, 1987, pp. 137–151. Expanded version available as Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, MIT, April 1987.

[LT89] N. Lynch and M. Tuttle, "An Introduction to Input/Output Automata," *CWI-Quarterly*, Vol. 2, No. 3, 1989. Also: Technical Memo, MIT/LCS/TM-373, Laboratory for Computer Science Massachusetts Institute of Technology, November 1988.

[M89] M. Merritt, "Completeness Theorems for Automata," REX Workshop, May 1989.

[MMT88] M. Merritt, F. Modugno and M. Tuttle "Time Constrained Automata," manuscript, November 1988.

[PF77] G. Peterson and M. Fischer, "Economical Solutions for the Critical Section Problem in a Distributed System," in *Proc. 9th ACM symp. on Theory of Computing*, May 1977, pp. 91–97.

[S88] F. B. Schneider, "Real-Time Reliable Systems Project," in *Foundations of Real-Time Computing Research Initiative*, ONR Kickoff Workshop, November 1988, pp. 28–32.

[SL87] A. U. Shankar and S. Lam, "Time-Dependent Distributed Systems: Proving Safety, Liveness and Timing Properties," *Distributed Computing*, 2 (1987), pp. 61–79.

[S89] A. C. Shaw, "Reasoning About Time in Higher-Level Language Software," *IEEE Transactions on Software Engineering*, Vol. SE-15, No. 7 (July 1989), pp. 875–889.

[SR89] J. Stankovic and K. Ramamritham, "The SPRING Kernel: A New Paradigm for Real-Time Operating Systems," *ACM Operating Systems Reviews*, Vol 23, No. 3 (July 1989), pp. 54–71.

[T88] G. Tel, "Assertional Verification of a Timer Based Protocol," in *Proc. ICALP '88*, Lecture Notes in Computer Science 317, Springer-Verlag, pp. 600–614.

[Zw88] A. Zwarico, *Timed Acceptance: an Algebra of Time Dependent Computing*, Ph.D. thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1988.

A Proofs of Lemmas

A.1 Proof of Lemma 4.1

Proof: By induction on the length of an execution leading to s. If the length $= 0$, then $s.TIMER = k > 0$, so the conditions are easily seen to be true. So suppose that $(s', (\pi, t), s)$ is a step of $time(A, b)$, where s' is reachable in n steps and the conditions are true for s'. We consider cases.

Case 1: $\pi = GRANT$.
Then the effect of $GRANT$ implies that $s.TIMER = k > 0$, which implies both conditions.

Case 2: $\pi = ELSE$.
The precondition of $ELSE$ implies that $s'.TIMER > 0$. Since $s.TIMER = s'.TIMER$, we also have $s.TIMER > 0$, which implies both conditions.

Case 3: $\pi = TICK$.
Suppose that $s.TIMER < 0$, Then $s'.TIMER = 0$, by the inductive hypothesis. The inductive hypothesis also implies that $s'.Ft(TICK) \geq s'.Lt(LOCAL) + c_1 - l$. Since $c_1 > l$ (by an assumption), this implies that $s'.Ft(TICK) > s'.Lt(LOCAL)$. But then $TICK$ is not enabled in s', a contradiction. Thus, $s.TIMER \geq 0$, showing the first property.

Now, $s.Ft(TICK) = t + c_1$ and $s.Lt(LOCAL) \leq t + l$. This implies that

$$s.Ft(TICK) \geq s.Lt(LOCAL) + c_1 - l,$$

showing the second property. ■

A.2 Proof of Lemma 4.4

Proof: We begin by giving an explicit description of B, by instantiating the general definition of $time(A, \mathcal{U})$ for the case where $\mathcal{U}$ is the given set of conditions. We use this explicit description in the proof below.

Each state of B has components As, holding a state of A, plus Ct, $Ft(G_1)$, $Lt(G_1)$, $Ft(G_2)$ and $Lt(G_2)$. Each initial state of B consists of an initial state s of A, plus $Ct = 0$, plus $Ft(G_1) = k \cdot c_1$, $Lt(G_1) = k \cdot c_2 + l$, $Ft(G_2) = 0$ and $Lt(G_2) = \infty$. If (π, t) is an action of B, then $(s', (\pi, t), s)$ is a step of B exactly if the following conditions hold.

1. $(s'.As, \pi, s.As)$ is a step of A.
2. $s'.Ct \leq t = s.Ct$.
3. If $\pi = GRANT$ then
   (a) $s'.Ft(G_1) \leq t \leq s'.Lt(G_1)$ and $s'.Ft(G_2) \leq t \leq s'.Lt(G_2)$,

(b) $s.Ft(G_2) = t + k \cdot c_1 - l$ and $s.Lt(G_2) = t + k \cdot c_2 + l$,

(c) $s.Ft(G_1) = 0$ and $s.Lt(G_2) = \infty$.

4. If $\pi = ELSE$ or $TICK$, then

(a) $t \leq s'.Lt(G_1)$ and $t \leq s'.Lt(G_2)$,

(b) $s.Ft(G_1) = s'.Ft(G_1)$, $s.Lt(G_1) = s'.Lt(G_1)$, $s.Ft(G_2) = s'.Ft(G_2)$, and $s.Lt(G_2) = s'.Lt(G_2)$.

Let s and u be the unique start states of $time(A,b)$ and B, respectively. Then $s.TIMER = k > 0$. Also,

$$\min(u.Lt(G_1), u.Lt(G_2)) = k \cdot c_2 + l \quad \text{and} \quad s.Lt(TICK) = c_2.$$

It follows that

$$\min(u.Lt(G_1), u.Lt(G_2)) = s.Lt(TICK) + (s.TIMER - 1)c_2 + l.$$

Furthermore,

$$\max(u.Ft(G_1), u.Ft(G_2)) = k \cdot c_1 \quad \text{and} \quad s.Ft(TICK) = c_1,$$

so that

$$\max(u.Ft(G_1), u.Ft(G_2)) = s.Ft(TICK) + (s.TIMER - 1)c_1.$$

This suffices to show the initial condition.

Now consider a step $(s', (\pi, t), s)$ of $time(A,b)$, where s' is a reachable state of $time(A,b)$, and suppose that u' is a reachable state of B such that $u' \in f(s')$. We argue that (π, t) is enabled in u'. The first thing we must show is that

$$t \leq \min(u'.Lt(G_1), u'.Lt(G_2)).$$

If this is not the case, then

$$t > \min(u'.Lt(G_1), u'.Lt(G_2)).$$

Since s' is a reachable state of $time(A,b)$, Lemma 4.1 implies that $s'.TIMER \geq 0$. Then since $u' \in f(s')$, it follows that either

$$\min(u'.Lt(G_1), u'.Lt(G_2)) \geq s'.Lt(TICK)$$

$$\text{or} \quad \min(u'.Lt(G_1), u'.Lt(G_2)) \geq s'.Lt(LOCAL).$$

Therefore, either

$$t > s'.Lt(TICK) \text{ or } t > s'.Lt(LOCAL).$$

Either case contradicts the operation of $time(A,b)$.

The other thing we must show is that if $\pi = GRANT$, then

$$\max(u'.Ft(G_1), u'.Ft(G_2)) \leq t\ .$$

Since $(GRANT, t)$ is enabled in s', it must be that $s'.TIMER \leq 0$, and Lemma 4.1 then implies that $s'.TIMER = 0$. Since $u' \in f(s')$, we have

$$\max(u'.Ft(G_1), u'.Ft(G_2)) \leq s'.Ct\ .$$

This means that

$$\max(u'.Ft(G_1), u'.Ft(G_2)) \leq s'.Ct \leq s.Ct = t,$$

as needed.

Now we consider cases.

Case 1: $\pi = GRANT$. Then define u so that

$$u.Ft(G_1) = 0,$$

$$u.Lt(G_1) = \infty,$$

$$u.Ft(G_2) = t + k \cdot c_1 - l \text{ and}$$

$$u.Lt(G_2) = t + k \cdot c_2 + l.$$

(Other components are exactly as in s.) The preconditions already checked imply that $(u', (\pi, t), u)$ is a step of B. It remains to show that $u \in f(s)$. The effects of the $GRANT$ action imply that $s.TIMER = k > 0$. Thus, we must show that

$$\min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(TICK) + (s.TIMER - 1)c_2 + l$$

and

$$\max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ft(TICK) + (s.TIMER - 1)c_1.$$

To see the first inequality, note that

$$s.Lt(TICK) \leq t + c_2;$$

thus,

$$\begin{aligned} s.Lt(TICK) \ &+ \ (s.TIMER - 1)c_2 + l \\ &\leq \ t + c_2 + (k-1)c_2 + l \\ &= \ t + k \cdot c_2 + l, \end{aligned}$$

which shows the first inequality.

To see the second inequality, note that $\max(u.Ft(G_1), u.Ft(G_2)) = t + k \cdot c_1 - l$. The definition of $time(A, b)$ implies that

$$s.Ct \leq s'.Lt(LOCAL).$$

Lemma 4.1 implies that

$$s'.Ft(TICK) \geq s'.Lt(LOCAL) + c_1 - l.$$

Therefore,

$$\begin{aligned} s.Ft(TICK) + (s.TIMER - 1)c_1 &= s'.Ft(TICK) + (s.TIMER - 1)c_1 \\ &\geq s'.Lt(LOCAL) + c_1 - l + (s.TIMER - 1)c_1 \\ &\geq t + c_1 - l + (s.TIMER - 1)c_1 \\ &= t + k \cdot c_1 - l, \end{aligned}$$

which implies the second inequality.

Case 2: $\pi = ELSE$. Then define u so that

$$u.Ft(G_1) = u'.Ft(G_1),$$

$$u.Lt(G_1) = u'.Lt(G_1),$$

$$u.Ft(G_2) = u'.Ft(G_2), \text{ and}$$

$$u.Lt(G_2) = u'.Lt(G_2).$$

(Other components are exactly as in s.) The preconditions already checked imply that $(u', (\pi, t), u)$ is a step of B. It remains to show that $u \in f(s)$. Since $(ELSE, t)$ is enabled in $time(A, b)$, we have $s'.TIMER > 0$. Since $s.TIMER = s'.TIMER$, we also have $s.TIMER > 0$. Thus, we must show that

$$\min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(TICK) + (s.TIMER - 1)c_2 + l,$$

and

$$\max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ft(TICK) + (s.TIMER - 1)c_1.$$

To see the first inequality, note that the inductive hypothesis implies that

$$\min(u'.Lt(G_1), u'.Lt(G_2)) \geq s'.Lt(TICK) + (s'.TIMER - 1)c_2 + l.$$

But

$$\min(u.Lt(G_1), u.Lt(G_2)) = \min(u'.Lt(G_1), u'.Lt(G_2)),$$

and

$$s.Lt(TICK) = s'.Lt(TICK).$$

Therefore, the first inequality holds.

To see the second inequality, note that the inductive hypothesis implies that

$$\max(u'.Ft(G_1), u'.Ft(G_2)) \leq s'.Ft(TICK) + (s'.TIMER - 1)c_1.$$

But

$$\max(u.Ft(G_1), u.Ft(G_2)) = \max(u'.Ft(G_1), u'.Ft(G_2)),$$

and

$$s.Ft(TICK) = s'.Ft(TICK).$$

Therefore, the second inequality holds.

Case 3: $\pi = TICK$. Then define u so that

$$u.Ft(G_1) = u'.Ft(G_1),$$

$$u.Lt(G_1) = u'.Lt(G_1),$$

$$u.Ft(G_2) = u'.Ft(G_2), \text{ and}$$

$$u.Lt(G_2) = u'.Lt(G_2).$$

(Other components are exactly as in s.) The preconditions already checked imply that $(u', (\pi, t), u)$ is a step of B. It remains to show that $u \in f(s)$. Note that $s.TIMER = s'.TIMER - 1$. There are two subcases to consider.

1. $s.TIMER > 0$.

   Then we must show that

   $$\min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(TICK) + (s.TIMER - 1)c_2 + l,$$

   and

   $$\max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ft(TICK) + (s.TIMER - 1)c_1.$$

   To see the first inequality, note that the inductive hypothesis implies that

   $$\min(u'.Lt(G_1), u'.Lt(G_2)) \geq s'.Lt(TICK) + (s'.TIMER - 1)c_2 + l.$$

   But

   $$\min(u.Lt(G_1), u.Lt(G_2)) = \min(u'.Lt(G_1), u'.Lt(G_2)),$$

   $$s.Lt(TICK) = t + c_2,$$

   and

   $$t \leq s'.Lt(TICK).$$

   Therefore, we have

$$\begin{aligned} min(u.Lt(G_1), u.Lt(G_2)) &= \min(u'.Lt(G_1), u'.Lt(G_2)) \\ &\geq s'.Lt(TICK) + (s'.TIMER - 1)c_2 + l \\ &= s'.Lt(TICK) + ((s.TIMER + 1) - 1)c_2 + l \\ &= c_2 + s'.Lt(TICK) + (s.TIMER - 1)c_2 + l \\ &= s.Lt(TICK) - t + s'.Lt(TICK) + (s.TIMER - 1)c_2 + l. \end{aligned}$$

Since the definition of $time(A, b)$ implies that

$$s'.Lt(TICK) \geq t,$$

the first inequality follows.

To see the second inequality, note that the inductive hypothesis implies that

$$\max(u'.Ft(G_1), u'.Ft(G_2)) \leq s'.Ft(TICK) + (s'.TIMER - 1)c_1.$$

But

$$\max(u.Ft(G_1), u.Ft(G_2)) = \max(u'.Ft(G_1), u'.Ft(G_2)),$$

and

$$s.Ft(TICK) = t + c_1.$$

Furthermore, by the definition of $time(A, b)$,

$$t \geq s'.Ft(TICK).$$

Hence,

$$\begin{aligned}
\max(u.Ft(G_1), u.Ft(G_2)) &= \max(u'.Ft(G_1), u'.Ft(G_2)) \\
&\leq s'.Ft(TICK) + (s'.TIMER - 1)c_1 \\
&= s'.Ft(TICK) + ((s.TIMER + 1) - 1)c_1 \\
&= c_1 + s'.Ft(TICK) + (s.TIMER - 1)c_1 \\
&\leq c_1 + t + (s.TIMER - 1)c_1 \\
&= s.Ft(TICK) + (s.TIMER - 1)c_1,
\end{aligned}$$

as needed.

2. $s.TIMER = 0$.

Then we must show that

$$\min(u.Lt(G_1), u.Lt(G_2)) \geq s.Lt(LOCAL)$$

and

$$\max(u.Ft(G_1), u.Ft(G_2)) \leq s.Ct.$$

Note that $s'.TIMER = 1$.

To see the first claim, note that $s'.TIMER > 0$, so the inductive hypothesis implies that

$$\begin{aligned}
\min(u'.Lt(G_1), u'.Lt(G_2)) &\geq s'.Lt(TICK) + (s'.TIMER - 1)c_2 + l \\
&= s'.Lt(TICK) + l
\end{aligned}$$

Furthermore, note that the definition of $time(A,b)$ implies that

$$t \leq s'.Lt(TICK).$$

Hence

$$\begin{aligned} \min(u.Lt(G_1), u.Lt(G_2)) &= \min(u'.Lt(G_1), u'.Lt(G_2)) \\ &\geq s'.Lt(TICK) + l. \\ &\geq s'.Ct + l \\ &\geq s.Lt(LOCAL), \end{aligned}$$

which shows the first claim.

To see the second claim, note that $s'.TIMER > 0$, so the inductive hypothesis implies that

$$\begin{aligned} \max(u'.Ft(G_1), u'.Ft(G_2)) &\leq s'.Ft(TICK) + (s'.TIMER - 1)c_1 \\ &= s'.Ft(TICK). \end{aligned}$$

Now, $s'.Ft(TICK) \leq t$, so that

$$\max(u'.Ft(G_1), u'.Ft(G_2)) \leq t.$$

But

$$\begin{aligned} \max(u.Ft(G_1), u.Ft(G_2)) &= \max(u'.Ft(G_1), u'.Ft(G_2)) \\ &\leq t = s.Ct\ , \end{aligned}$$

as needed.

■

A.3 Proof of Lemma 5.6

Proof: We begin by giving an explicit description of B_k, by instantiating the general definition of $time(A',\mathcal{U})$ for the case where $\mathcal{U} = \mathcal{U}_k$. We use this explicit description in the proof below.

Each state of B_k has component As, holding a state of A', plus Ct, $Ft(k,n)$, $Lt(k,n)$, $Ft(SIGNAL_i)$ and $Lt(SIGNAL_i)$, for every i, $0 \leq i \leq k$, $Ft(NULL)$ and $Lt(NULL)$. Each initial state of B_k consists of an initial state s of A', plus $Ct = 0$, $Ft(NULL) = n_1$, $Lt(NULL) = n_2$, all other Ft components equal to 0, and all other Lt components equal to ∞. If (π,t) is an action of B_k, then $(s',(\pi,t),s)$ is a step of B_k exactly if the following conditions hold.

1. $(s'.As, \pi, s.As)$ is a step of A.

2. $s'.Ct \leq t = s.Ct$.

3. $t \leq s'.Lt(k,n)$, $t \leq s'.Lt(SIGNAL_i)$ for all $i, 0 \leq i \leq k$, and $t \leq s'.Lt(NULL)$.

4. If $\pi = SIGNAL_i$, for $0 \leq i \leq k-1$, then

   (a) $s'.Ft(SIGNAL_i) \leq t$,

   (b) $s.Ft(SIGNAL_i) = 0$ and $s.Lt(SIGNAL_i) = \infty$,

   (c) $s.Ft(SIGNAL_{i+1}) = t + d_1$ and $s.Lt(SIGNAL_{i+1}) = t + d_2$, and

   (d) $s.Ft(k,n) = s'.Ft(k,n)$, $s.Lt(k,n) = s'.Lt(k,n)$, and $s.Ft(C) = s'.Ft(C)$ and $s.Lt(C) = s'.Lt(C)$ for all partition classes $C \notin \{SIGNAL_i, SIGNAL_{i+1}\}$.

5. If $\pi = SIGNAL_k$, then

   (a) $s'.Ft(SIGNAL_k) \leq t$,

   (b) $s.Ft(SIGNAL_k) = 0$ and $s.Lt(SIGNAL_k) = \infty$,

   (c) $s.Ft(k,n) = t + (n-k) \cdot d_1$ and $s.Lt(k,n) = t + (n-k) \cdot d_2$, and

   (d) $s.Ft(C) = s'.Ft(C)$ and $s.Lt(C) = s'.Lt(C)$ for all partition classes $C \neq SIGNAL_k$.

6. If $\pi = SIGNAL_i$, for $k+1 \leq i \leq n-1$, then

   (a) $s.Ft(k,n) = s'.Ft(k,n)$, $s.Lt(k,n) = s'.Lt(k,n)$, and $s.Ft(C) = s'.Ft(C)$ and $s.Lt(C) = s'.Lt(C)$ for all partition classes C.

7. If $\pi = SIGNAL_n$, then

   (a) $s'.Ft(k,n) \leq t$,

   (b) $s.Ft(k,n) = 0$ and $s.Lt(k,n) = \infty$.

   (c) $s.Ft(C) = s'.Ft(C)$ and $s.Lt(C) = s'.Lt(C)$ for all partition classes C.

8. If $\pi = NULL$ then

   (a) $s'.Ft(NULL) \leq t$,

   (b) $s.Ft(NULL) = n_1$ and $s.Lt(NULL) = n_2$, and

   (c) $s.Ft(k,n) = s'.Ft(k,n)$, $s.Lt(k,n) = s'.Lt(k,n)$, and $s.Ft(C) = s'.Ft(C)$ and $s.Lt(C) = s'.Lt(C)$ for all partition classes $C \neq NULL$.

The description of B_{k-1} is similar, but with $k-1$ replacing k. We now present the proof of Lemma 5.6. Let s and u be the unique start states of B_k and B_{k-1}, respectively. Then $u.Lt(k-1,n) = \infty$ and $u.Ft(k-1,n) = 0$, so the inequalities clearly hold, implying that $u \in f_k(s)$.

Now consider a step $(s', (\pi, t), s)$ of B_k, where s' is a reachable state of B_k, and suppose that u' is a reachable state of B_{k-1} such that $u' \in f_k(s')$. We first argue that (π, t) is enabled in u'.

There are two key facts that we must show. The first is that

$$t \leq u'.Lt(k-1,n).$$

The inductive hypothesis implies that:

$$u'.Lt(k-1,n) \geq \begin{cases} s'.Lt(k,n) & \text{if } s'.\text{FLAG}_i = true \\ & \text{for some } i, k+1 \leq i \leq n \\ s'.Lt(SIGNAL_k) + (n-k)d_2 & \text{if } s'.\text{FLAG}_k = true \\ \infty & \text{otherwise,} \end{cases}$$

First suppose that $u'.Lt(k-1,n) \geq s'.Lt(k,n)$; then since (π,t) is enabled in s', it must be that $t \leq s'.Lt(k,n)$. Thus, $t \leq u'.Lt(k-1,n)$ in this case. Second, suppose that $u'.Lt(k-1,n) \geq s'.Lt(SIGNAL_k) + (n-k)d_2 \geq s'.Lt(SIGNAL_k)$. Since (π,t) is enabled in s', it must be that $t \leq s'.Lt(SIGNAL_k)$. Therefore, $t \leq u'.Lt(k-1,n)$ in this case. The only remaining case is that $u'.Lt(k-1,n) = \infty$, in which case the condition clearly holds.

The second key fact to show is that if $\pi = SIGNAL_n$, then

$$t \geq u'.Ft(k-1,n).$$

So suppose that $\pi = SIGNAL_n$. Since π is enabled in s', it must be that $s'.\text{FLAG}_n = true$. Since $u' \in f_k(s')$, $s'.\text{FLAG}_k = true$, and $k < n$, the definition of f_k implies that

$$u'.Ft(k-1,n) \leq s'.Ft(k,n)$$

But $t \geq s'.Ft(k,n)$ since (π,t) is enabled in s'. Therefore, $t \geq u'.Ft(k-1,n)$, as needed.

Thus, (π,t) is enabled in u'. To complete the proof, we must show that (for s', u' and π as described above) there exists a state u of B_{k-1} such that $(u',(\pi,t),u)$ is a step of B_{k-1} and $u \in f_k(s)$. We define u to be the unique state defined by $u.As = s.As$ and Ft and Lt components as implied by the construction of B_{k-1}, such that $(u',(\pi,t),u)$ is a step of B_{k-1}; it remains to show that $u \in f_k(s)$. We consider cases.

Case 1: $\pi = SIGNAL_i$, for $0 \leq i \leq k-2$.

Then $u.Ft(k-1,n) = 0$ and $u.Lt(k-1,n) = \infty$, which immediately imply the inequalities. Also, since $u.Ft(SIGNAL_i) = s.Ft(SIGNAL_i) = 0$ and $u.Lt(SIGNAL_i) = s.Lt(SIGNAL_i) = \infty$, and all components of u' other than $u'.Ft(k-1,n)$ and $u'.Lt(k-1,n)$ have the same value as the corresponding components of s', it follows that all components of u other than $u.Ft(k-1,n)$ and $u.Lt(k-1,n)$ have the same value as the corresponding components of s. Therefore, $u \in f_k(s)$.

Case 2: $\pi = SIGNAL_{k-1}$.

Then

$$u.Lt(k-1,n) = t + (n-(k-1))d_2 = t + (n-k+1)d_2,$$

$$
\begin{aligned}
u.Ft(k-1,n) &= t + (n-(k-1))d_1 = t + (n-k+1)d_1, \\
s.Lt(SIGNAL_k) &= t + d_2, \\
s.Ft(SIGNAL_k) &= t + d_1, \\
\text{and} \quad s.\mathrm{FLAG}_k &= true.
\end{aligned}
$$

Thus we have

$$u.Lt(k-1,n) = t + (n-k+1)d_2 = t + d_2 + (n-k)d_2 = s.Lt(SIGNAL_k) + (n-k)d_2$$

and

$$u.Ft(k-1,n) = t + (n-k+1)d_1 = t + d_1 + (n-k)d_1 = s.Ft(SIGNAL_k) + (n-k)d_1.$$

This implies the inequalities. The equivalence of corresponding components of u and s is straightforward, as in Case 1.

Case 3: $\pi = SIGNAL_k$.

Then

$$
\begin{aligned}
u.Lt(k-1,n) &= u'.Lt(k-1,n), \\
u.Ft(k-1,n) &= u'.Ft(k-1,n), \\
s.Lt(k,n) &= t + (n-k)d_2, \\
s.Ft(k,n) &= t + (n-k)d_1, \\
s'.FLAG_k &= true, \\
\text{and} \quad s.\mathrm{FLAG}_{k+1} &= true.
\end{aligned}
$$

Since $s.FLAG_{k+1} = true$, the inequalities we need to show are:

$$u.Lt(k-1,n) \geq s.Lt(k,n) \quad \text{and} \quad u.Ft(k-1,n) \leq s.Ft(k,n).$$

For the upper bound,

$$
\begin{aligned}
u.Lt(k-1,n) &= u'.Lt(k-1,n) \\
&\geq s'.Lt(SIGNAL_k) + (n-k)d_2 \\
&\quad \text{since } u' \in f_k(s') \text{ and } s'.\mathrm{FLAG}_k = true, \\
&\geq t + (n-k)d_2 \\
&\quad \text{since } t \leq s'.Lt(SIGNAL_k) \text{ by the fact} \\
&\quad \text{that } (\pi, t) \text{ is enabled in } s', \\
&= s.Lt(k,n).
\end{aligned}
$$

For the lower bound we get, using similar reasoning,

$$\begin{aligned} u.Ft(k-1,n) &= u'.Ft(k-1,n) \\ &\leq s'.Ft(SIGNAL_k) + (n-k)d_1 \\ &\leq t + (n-k)d_1 \\ &= s.Ft(k,n). \end{aligned}$$

The equivalence of corresponding components of u and s is again straightforward.

Case 4: $\pi = SIGNAL_i$, for $k+1 \leq i \leq n-1$.

This step does not change any Ft or Lt component of either B_k or B_{k-1}. Thus, the inequalities and equivalences are all preserved.

Case 5: $\pi = SIGNAL_n$.

Then $u.Lt(k-1,n) = \infty$ and $u.Ft(k-1,n) = 0$, so that the inequalities are immediate; the equivalences are again straightforward.

Case 6: $\pi = NULL$.

This step does not change any of the Ft or Lt components involved in the inequalities, so that the inequalities are preserved. Since the only changes to Ft and Lt components made by this step are to set $u.Ft(NULL) = s.Ft(NULL) = t + n_1$ and $u.Lt(NULL) = s.Lt(NULL) = t + n_2$, the equivalences are again straightforward. ■

OFFICIAL DISTRIBUTION LIST

Director — 2 copies
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

Office of Naval Research — 2 copies
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. Gary Koop, Code 433

Director, Code 2627 — 6 copies
Naval Research Laboratory
Washington, DC 20375

Defense Technical Information Center — 12 copies
Cameron Station
Alexandria, VA 22314

National Science Foundation — 2 copies
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

Dr. E.B. Royce, Code 38 — 1 copy
Head, Research Department
Naval Weapons Center
China Lake, CA 93555